HW1: Dynamic Programming

Emma Clark
Department of Aerospace Engineering
UIUC
emmac4

I. Environments

A. Gridworld

This environment consists of a simple 5×5 grid; giving 25 possible states. There are four possible actions: (1) right, (2) up, (3) left, (4) down. The transition model of this environment is explicitly defined, and accessable by each algorithm; each state transition corresponds to a certain reward. (NOTE: the "easy" version of transition probabilities and rewards is what is being evaluated in this report). The algorithms will then be trained to find a trajectory that maximizes the episode reward.

B. Discrete Pendulum

This environment consists of a simple pendulum with a discretized state and action space. Contrary to the gridworld environment, the explicit model is not available. The states are functions of the angle θ displacement of the pendulum from the upright axis. There is a maximum allowed angle velocity $\dot{\theta}$ value; if this constraint is violated a large negative reward is received. There is a goal range for θ, that holds the pendulum in an upright position, for which a $+1$ reward is received. Otherwise, a reward of 0 is received. Therefore, the algorithms will be trained to keep the pendulum inverted for as many time-steps as possible during an episode.

II. Algorithms

A. Policy Iteration

Policy iteration is a model-based algorithm that consists of two stages: (1) Policy Evaluation and (2) Policy Improvement. Policy evaluation assesses how "good" the current policy π is by iteratively applying the Bellman Equation:

$$V_{k+1}^{\pi} = \sum_{a} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s,a)[r(s,a) + \gamma V_k^{\pi}(s')] \tag{1}$$

where γ is the discount factor. Each policy evaluation stage iterates through $|\mathcal{S}|$ updates. Following policy evaluation is policy improvement. Policy improvement establishes a greedy policy π':

$$\pi'(s) = \arg\max_{a} p(s'|s,a)[r(s,a) + \sum_{s' \in \mathcal{S}} \gamma V_k^{\pi}(s')] \tag{2}$$

This policy update satisfies the policy improvement theorem

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \quad \forall s \in \mathcal{S} \tag{3}$$

We apply Policy Iteration to both the (easy) gridworld and the pendulum environments.

B. Value Iteration

Value iteration is another model based policy that utilizes Policy Evaluation and Policy Improvement; however, Value Iteration truncates the policy evaluation to one step, and combines it with policy improvement:

$$V_{k+1}(s) = \max_{a} p(s'|s,a)[r(s,a) + \sum_{s' \in \mathcal{S}} \gamma V_k(s')] \tag{4}$$

where γ is the discount factor. We apply Policy Iteration to both the (easy) gridworld and the pendulum environments.

C. SARSA

SARSA (State, Action, Reward, State, Action) is a model-free algorithm. It uses an $\epsilon - greedy$ policy:

$$\pi(s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s_t)|}, & n\text{if } a = A^* \\ \frac{\epsilon}{|\mathcal{A}(s_t)|}, & \text{if } a \neq A^* \end{cases} \tag{5}$$

to select an action given a current state. Then this action is used to take a step in the environment and get the reward, then the next action is selected using the $\epsilon - greedy$ policy and the new state. These values are then used to calculate the action-value update:

$$Q'(s,a) = Q(s_t, a_t) + \alpha(R_{t+1} + \gamma Q(s_{t+1}, a_{t+1} - Q(s_t - a_t)) \tag{6}$$

where γ is the discount factor and α is a learning rate. The learned policy π^* is then given by:

$$\pi^*(s) = \arg\max_{a} Q(s,a) \tag{7}$$

We apply SARSA to the pendulum environment.

D. Q-Learning

Q-Learning is similarly a model-free policy; but it is an off-policy technique. This allows the algorithm to learn about the greedy policy while following an exploration policy. Similar to SARSA, Q-learning starts by taking an $\epsilon - greedy$ action, then taking a step in the environment to get a new state and reward. Then the action-value function is updated with:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{8}$$

where γ is the discount factor and α is a learning rate. The learned policy is again given by (7). We apply Q-learning to the pendulum environment.

E. TD(0)

TD(0) is a temporal difference method for learning a value function V^π from sampled episodes under policy π. TD learning uses bootstrapping to update an estimate based on an estimate, and updates the value function at every episode step using:

$$V(s) = V(s) + \alpha[R + \gamma V(s') - V(s)] \tag{9}$$

where $gamma$ is the discount factor and α is the learning rate. We apply this algorithm to learn the value functions under the learned SARSA and Q-Learning policies in the pendulum environment.

III. Results

A. Gridworld

The learning curve and sample trajectories of the Policy Iteration algorithm is shown in figures 1 and 2, respectively. The parameters used in training can be seen in table I. The value θ is a threshold that determines the accuracy of the value function estimation. The policy continues to iterate until $\Delta = \max(\Delta, |V' - V|)$ is less than θ.

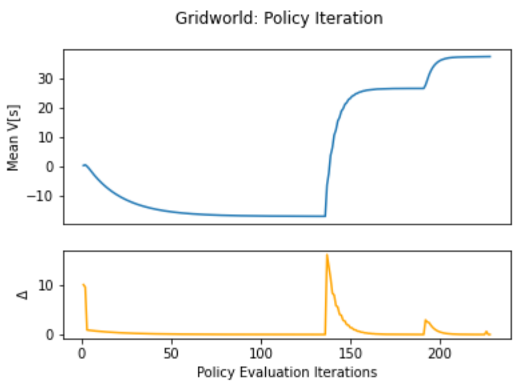

Fig. 1. Policy Iteration Training Curve

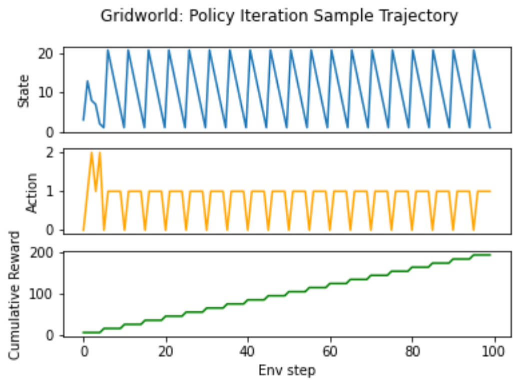

Fig. 2. Policy Iteration Sample Trajectory

The learning curve and sample trajectories of the Value Iteration algorithm is shown in figures 3 and 4, respectively. The parameters used in training can be seen in table I. The value θ and Δ represent the same thing as in Policy Iteration.

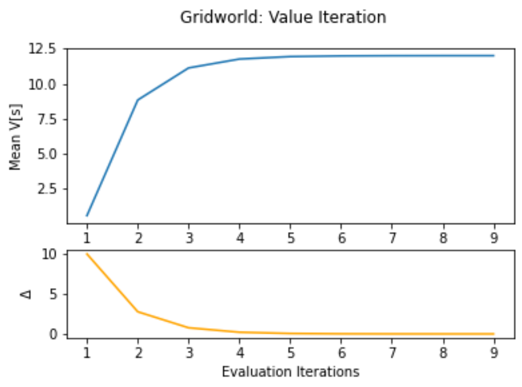

Fig. 3. Value Iteration Training Curve

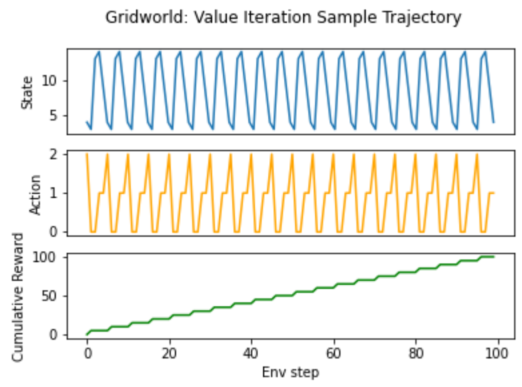

Fig. 4. Value Iteration Sample Trajectory

For SARSA, we test several different values for α and ϵ. During training in both algorithms ϵ has exponential decay; meaning, every 10 iterations the value of ϵ is decayed such that it may satisfy a greedy in the limit condition for convergence to an optimal policy. Examining the the plots for the different

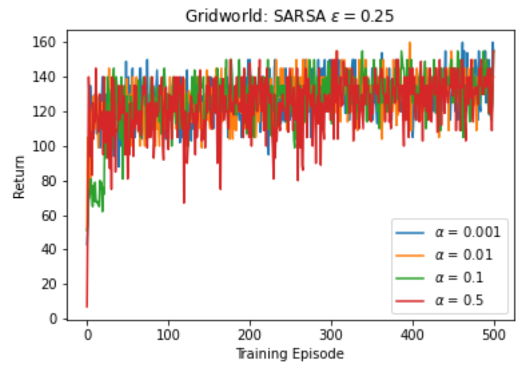

Fig. 5. SARSA: training with different α values

α and ϵ value in figures 5 and 6, we choose the values that give the most promising results to get a policy π. These parameters

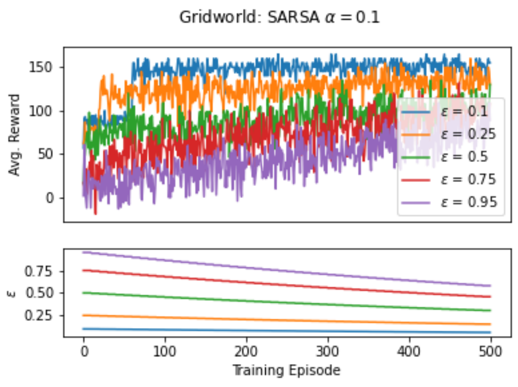

Fig. 6. SARSA: training with different ϵ values

are seen in Table I; the sample trajectory for this policy can be shown in figure 7.

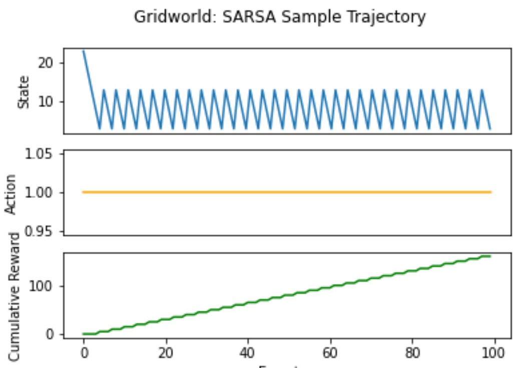

Fig. 7. SARSA sample trajectory

We perform the same parameter search for Q-Learning as we did for SARSA. Figures 8 and 9 show the plots of several training iterations for various α and ϵ values. Again, based on these results, we choose a set of parameters, shown in Table I, to get a policy for future plots. Figure 10 shows the trajectory of this policy.

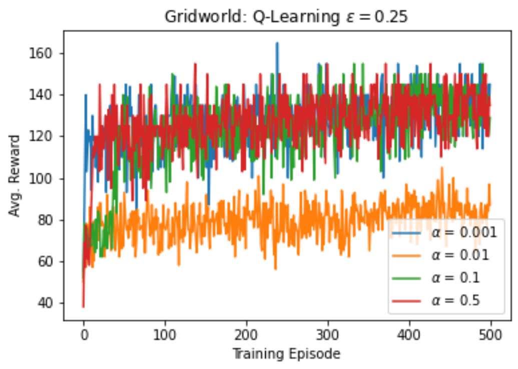

Fig. 8. Q-Learning: training with different α values

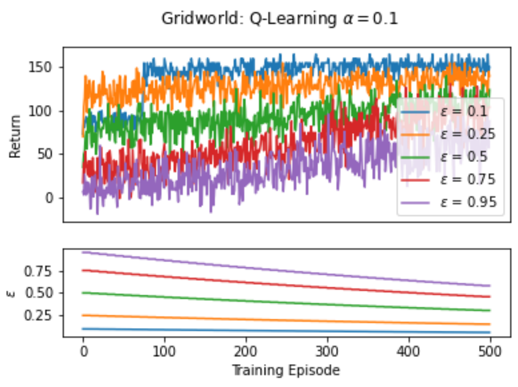

Fig. 9. Q-Learning: training with different ϵ values

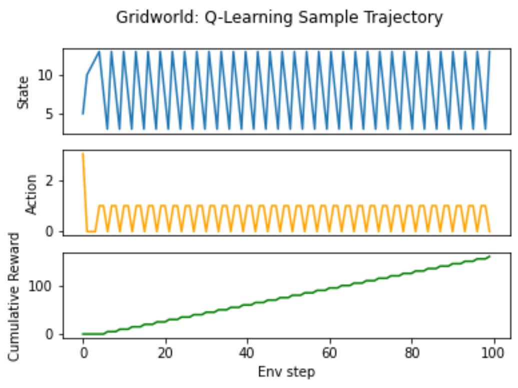

Fig. 10. Q-Learning sample trajectory

Finally, we plot the action chose by each policy (trained from each algorithm) at each state in figure 11 and the value functions learned in Value Iteration and Policy Iteration, as well as the value function learned using TD(0) applied to the policies learned from SARSA and Q-Learning, are plotted in figure 12. The parameters used in the TD(0) training are shown in Table I, and is trained for 300 iterations to ensure it converges to the true value of each policy.

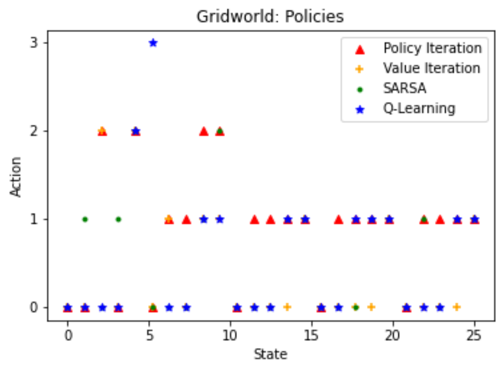

Fig. 11. Policies learned by each algorithm

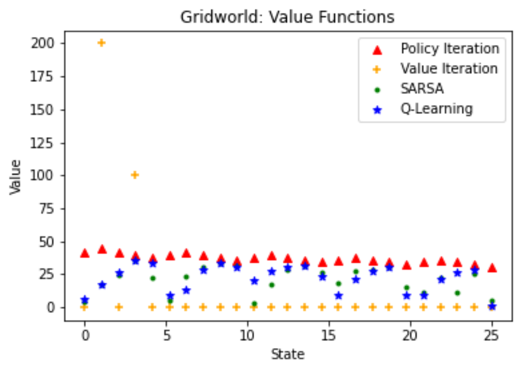

Fig. 12. Value functions learned from Policy and Value Iteration, and TD(0) [SARSA and Q-Learning Policies]

TABLE I
GRIDWORLD: FINAL PARAMETERS CHOSEN FOR EACH ALGORITHM

Algorithm	Parameters			
	γ	α	ϵ	θ
Policy Iteration	0.95	N/A	N/A	0.001
Value Iteration	0.95	N/A	N/A	0.001
SARSA	0.95	0.1	0.25	N/A
Q-Learning	0.95	0.1	0.25	N/A
TD(0)	0.95	0.1	N/A	N/A

B. Pendulum

We follow the same process used in the Gridworld environment to evaluate the SARSA, Q-Learning, and TD(0) algorithms. Figures 13 and 14 show the results of training SARSA with different α and ϵ values. The final training

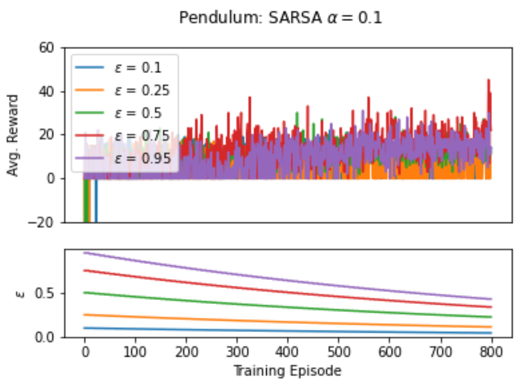

Fig. 13. SARSA: training with different α values

values to get our policy used later are shown in Table II. The trajectory of this policy is shown in 15.

We do this again for Q-Learning; shown in figures 16 and 17 are the results of the training on different α and ϵ values. We choose a final set of parameters, shown in Table II to get a policy, the trajectory of which is shown in figure 18.

Finally, we plot the policies and value functions (learned via TD(0)) for SARSA and Q-Learning in figures 19 and 20.

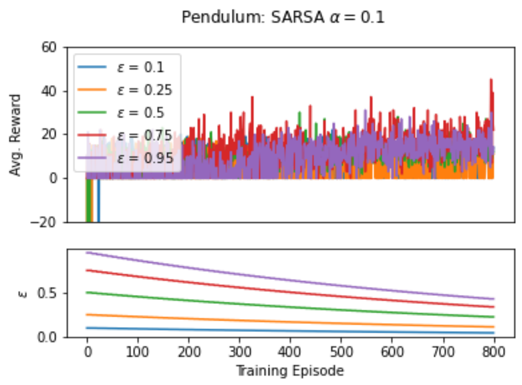

Fig. 14. SARSA: training with different ϵ values

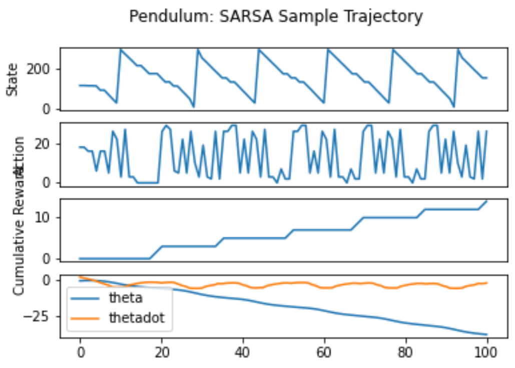

Fig. 15. SARSA sample trajectory

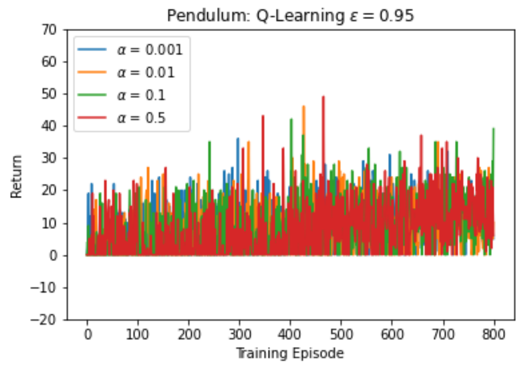

Fig. 16. Q-Learning: training with different α values

TABLE II
PENDULUM: FINAL PARAMETERS CHOSEN FOR EACH ALGORITHM

Algorithm	Parameters		
	γ	α	ϵ
SARSA	0.95	0.1	0.95
Q-Learning	0.95	0.1	0.95
TD(0)	0.95	0.1	N/A

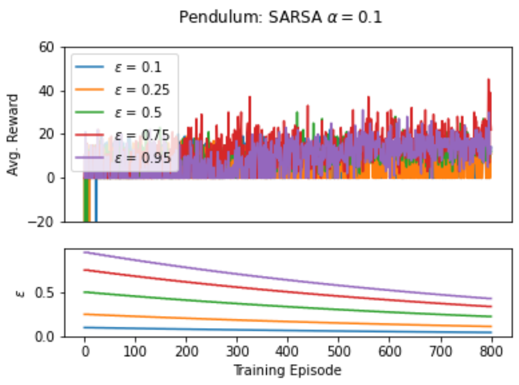

Fig. 17. Q-Learning: training with different ϵ values

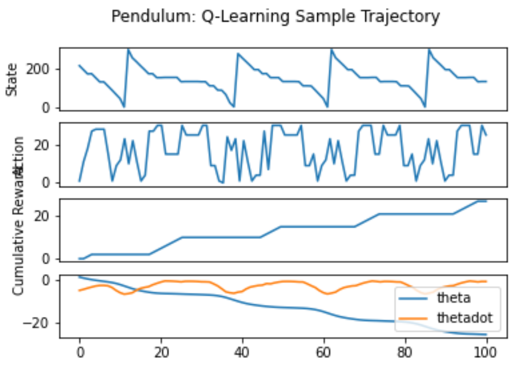

Fig. 18. Q-Learning sample trajectory

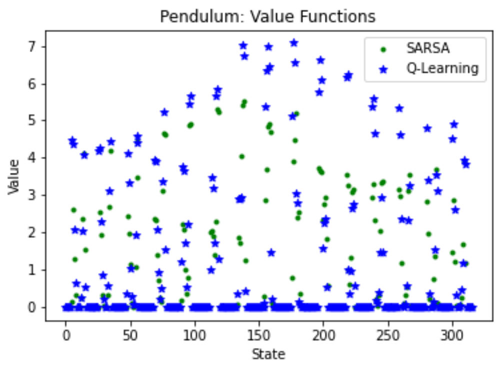

Fig. 20. Value functions learned from Policy and Value Iteration, and TD(0) [SARSA and Q-Learning Policies]

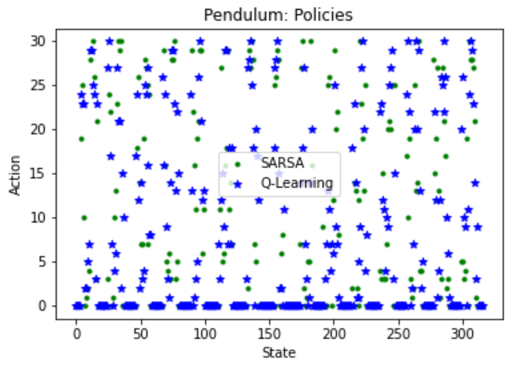

Fig. 19. Policies learned by each algorithm