Reinforcement Learning Homework 1

David M. Stanley (DavidMS4)*
University of Illinois, Urbana-Champaign, IL 61801

I. Nomenclature

In the interest of time, not all figures were formatted with TeX, so expect to see notations like 'α' and 'alpha' used interchangeably.

α/alpha	=	Learning rate $\in (0, 1]$
ϵ/epsilon	=	Exploration rate $\in [0, 1]$
γ/gamma	=	Successor state discount = 0.95 for all algorithms
π/pi	=	Policy
V	=	Value Function
s	=	State
s'	=	Successor state
a	=	Action
a'	=	Successor action
p	=	Transition probability
r	=	Reward
Q	=	State action pair value
$\mathcal{A}(s)$	=	Action space for each state s
$\mathcal{S}$	=	State space

II. Introduction

A. Gridworld

The gridworld model is shown in fig 1. It is a 5x5 Markov decision process model and in each state the agent is able to take 4 distinct actions: Move right, up, left & down. Any action taken in states A or B will move the agent to states A' or B' respectively and provide a reward of 10 or 5 respectively. If the agent takes an action that would move off the gridworld, then the agent remains in its current state and receives a reward of -1. An optimal policy and its corresponding value function can be found for this model using Policy Iteration and Value Iteration as the model is explicitly defined (and small enough to not be computationally intractable). In addition, SARSA and Q Learning are used to solve for the Q, state-action pair values. An optimal policy can be chosen from these values and the corresponding value function may be calculated using the temporal difference method (TD0).

B. Discrete Pendulum

The discrete pendulum model does not have an explicit model that may be accessed by Policy Iteration or Value Iteration so they are unusable for learning an optimal policy. SARSA and Q Learning are able to learn model free. For each action an integrator is used to predict what the next state for the pendulum will be. The continuous state is translated to a discretized set of angles and angle rates for the pendulum. The number of angle and angle rate states may be freely chosen. In this homework, 15 angles and 21 angle rates were chosen to discretize the state into 315 states. In addition, the action space may be discretized freely. 31 actions were chosen for this homework giving 9765 state-action pairs. If the pendulum is near the unstable equilibrium point, then the reward is 1 regardless of the pendulum's speed. If the magnitude of the pendulum's angular rate exceeds a limiting value, then a large negative reward of -100 is given.

*PhD Student, Department of Aerospace Engineering

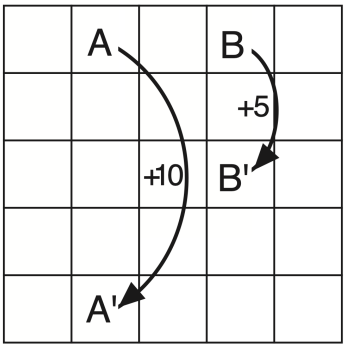

Fig. 1 5x5 Gridworld from Example 3.5, Chapter 3.5, Sutton and Barto

C. Explicit Model Algorithms

1. Policy Evaluation

Policy evaluation estimates the value function v_π for a given policy π and is used in Policy Iteration. The algorithm uses the Bellman equation to iteratively converge to the true value function of the given policy.

Algorithm 1 Policy Evaluation

Input: π
Parameters: $1 >> \theta > 0$
Initialization: $V[s] \leftarrow$ arbitrary $v \in \mathcal{R}\ \forall s \in \mathcal{S}$

$\Delta \leftarrow \infty$
while $\Delta \geq \theta$ **do**
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \Sigma_a \pi(a|s) \Sigma_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end for
end while

2. Policy Iteration

Policy Iteration makes use of policy evaluation to iteratively improve the policy and value function using the Bellman equation until the the policy is stable.

3. Value Iteration

Value Iteration is very similar to Policy Iteration, except rather than converging the value function to the true value for each policy improvement step, the policy evaluation only performs one iteration before choosing a new set of optimal actions. The optimal actions are not stored as a policy until the value function has converged.

D. Model Free Algorithms

1. TD(0) for predicting V(s)

Temporal Difference Prediction is an algorithm for immediately forming a target to estimate the current value of a function rather than waiting until the end of an episode to evaluate the discounted return for the step. We use it here for estimating the value function of a policy. TD(0) is a bootstrapping method because it uses an estimate of the next step's

Algorithm 2 Policy Iteration

Initialization: Assign $V(s) \leftarrow$ arbitrary $v \in \mathcal{R}$; and $\pi(s) \leftarrow$ arbitrary $a \in \mathcal{A}(s)\ \forall s \in \mathcal{S}$

$policy - stable \leftarrow false$
while $policy - stable = false$ **do**
    Policy Evaluation: see Algorithm 1

    Policy Improvement
    $policystable \leftarrow true$
    for each $s \in \mathcal{S}$ **do**
        $\pi_o \leftarrow \pi(s)$
        $\pi(s) \leftarrow argmax_a \Sigma_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        If $\pi_o \neq \pi(s)$, then $policy - stable \leftarrow false$
    end for
end while

Algorithm 3 Value Iteration

Input: $1 >> \theta > 0$
Initialization: $V(s) \leftarrow$, arbitrary $v \in \mathcal{R}, \forall s \in \mathcal{S}$
$\Delta \leftarrow \infty$
while $\Delta \geq \theta$ **do**
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow max_a \Sigma_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
    end for
end while
$\pi(s) \leftarrow argmax_a \Sigma_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

value to form an estimate for the current value. The advantage of this is that updates do not need to keep track of an entire episode. However, by using an estimate as the target, the current step inherits a bias from the initial value's guess. TD(0) is the basis for SARSA and Q Learning.

Algorithm 4 TD(0) for predicting $V(s)$

Input: π
Parameters: α
Initialization: $V(s) \leftarrow$, arbitrary $v \in \mathcal{R}, \forall s \in \mathcal{S}$

for each episode **do**
  $s \leftarrow s_{init}$
  for each step in episode **do**
    $a \leftarrow \pi(s)$
    Take action a and observe r and s'
    $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $s \leftarrow s'$
  end for
end for

2. SARSA

SARSA is a model free algorithm that learns state-action pair values. SARSA simply requires the current state, action and reward and the subsequent state and action to estimate Q(s,a). The subsequent state and action are used to bootstrap the learning process by setting $[r + \gamma Q(s', a')]$ as the TD target. In this homework an ϵ-greedy policy is used to explore the state-action space.

Algorithm 5 SARSA

Parameters: α, ϵ
Initialization: $Q(s, a) \leftarrow$, arbitrary $q \in \mathcal{R}, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$

for each episode **do**
  $s \leftarrow s_{init}$
  $a \leftarrow \epsilon GreedyAction(Q(s))$
  for each step in episode **do**
    Take action a and observe r and s'
    $a' \leftarrow \epsilon GreedyAction(Q(s'))$
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end for
end for

3. Q Learning

Like SARSA Q Learning is a model free algorithm that learns state-action pair values. The major difference is that while SARSA is on policy and always uses the same policy to choose the first action and subsequent action for bootstrapping, Q Learning always chooses the greediest action for the subsequent action. This makes Q Learning an offline learning algorithm because it is attempting to evaluate one policy while following another. In general Q Learning will find the most optimal path for a greedy policy even if its riskier under exploration.

Algorithm 6 Q Learning

Parameters: α, ϵ
Initialization: $Q(s,a) \leftarrow$, arbitrary $q \in \mathcal{R}, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$

for each episode **do**
  $s \leftarrow s_{init}$
  for each step in episode **do**
    $a \leftarrow \epsilon GreedyAction(Q(s))$
    Take action a and observe r and s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_a Q(s',a) - Q(s,a)]$
    $s \leftarrow s'$
  end for
end for

III. Results

A. Gridworld

The Gridworld is a fairly simple model. The most optimal policy is to move towards state B if starting in B or just to the right of state B, and to move to state A as quickly as possible in all other states.

1. Policy Iteration

Policy Iteration converges to the true value function and optimal policy after 5 iterations of policy imporvement. The optimal policy is to move to state A and unless starting in state B or to the right of state B.

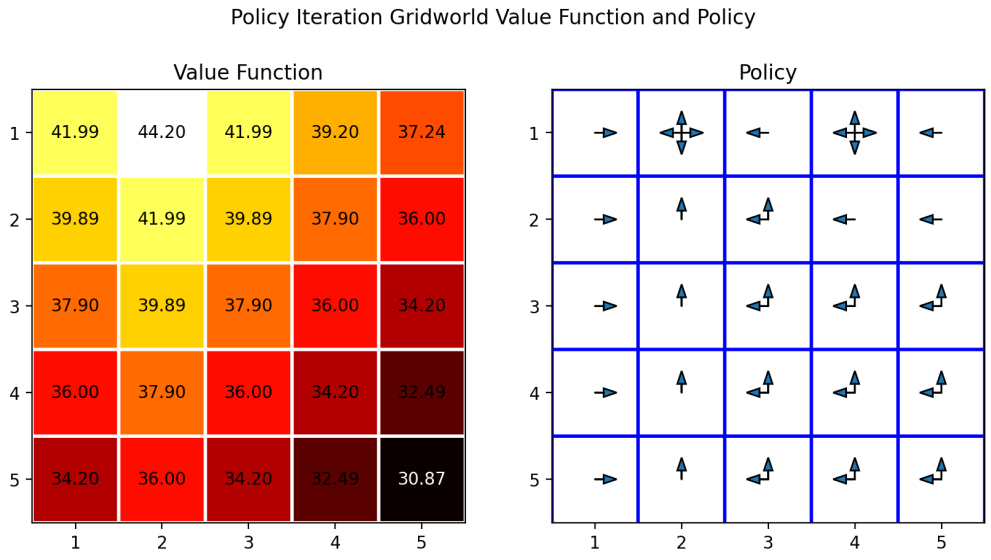

Fig. 2 Policy Iteration value function and policy

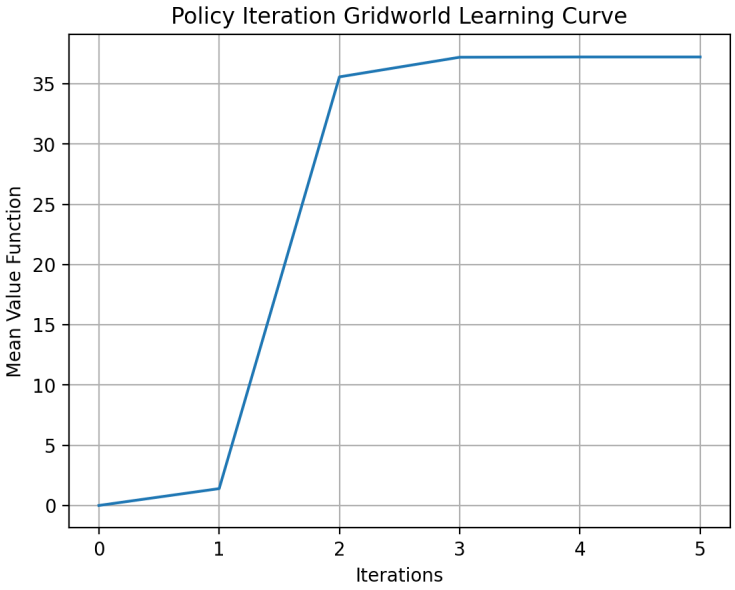

Fig. 3 Policy Iteration learning rate

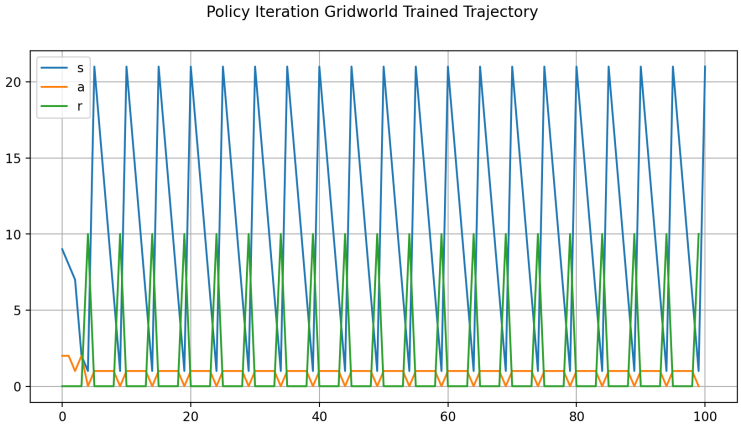

Fig. 4 Policy Iteration example trajectory

2. Value Iteration

Value Iteration converges to the true value function after 37 iterations. The policy derived from this value function is also optimal. The learning curve is much smoother than Policy Iteration because PI fully evaluates the value function for every policy iteration step which causes large jumps. In VI, since there is only one policy evaluation per change in actions, the learning rate is more gradual per iteration, but each iteration is faster.

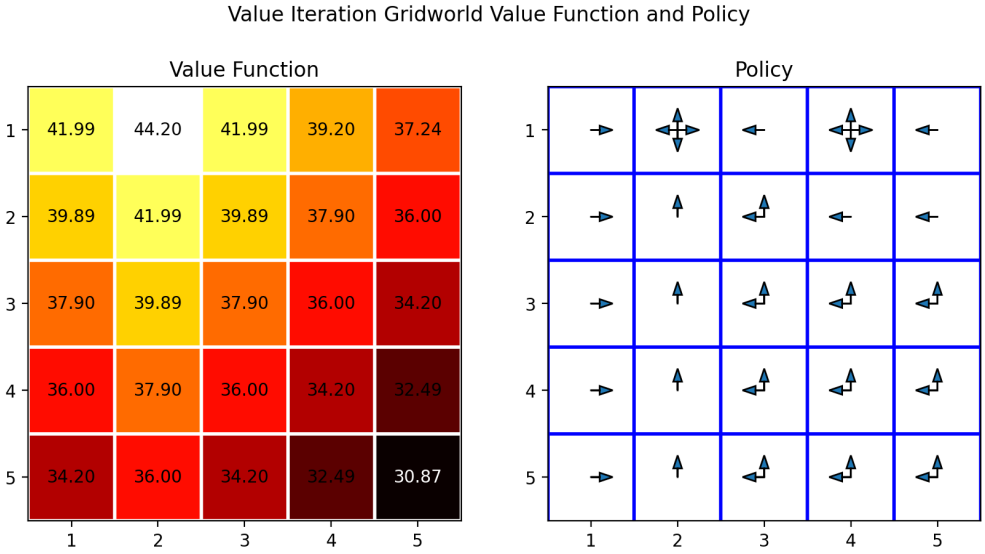

Fig. 5 Value Iteration value function and policy

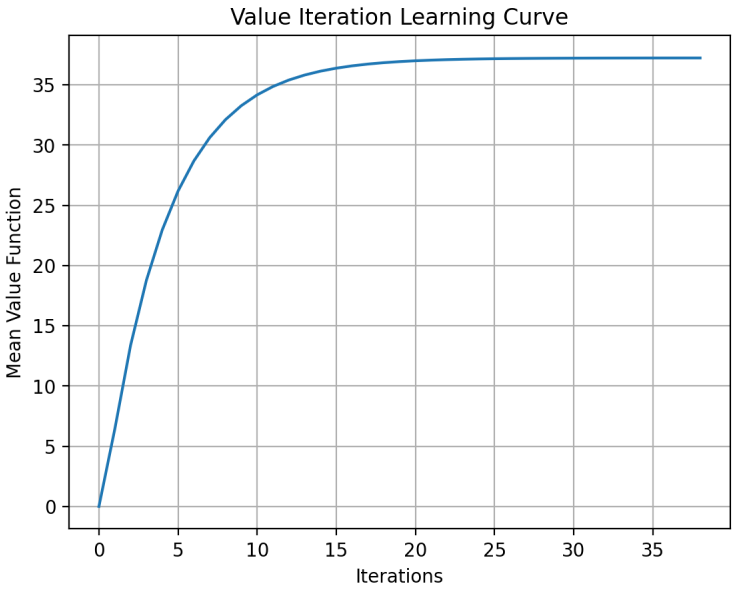

Fig. 6 Value Iteration learning rate

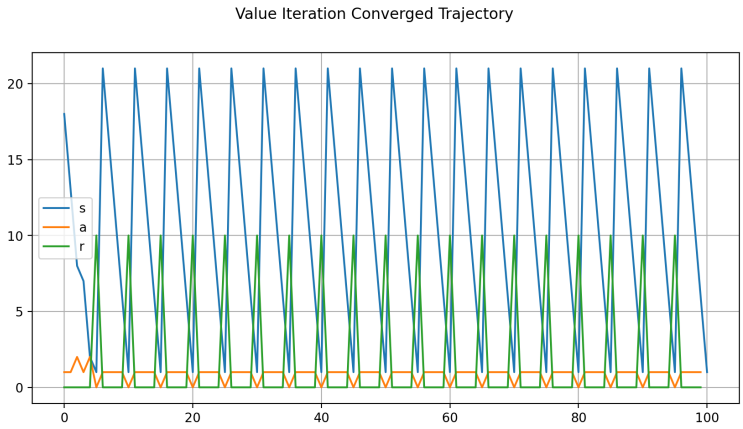

Fig. 7 Value Iteration example trajectory

3. SARSA

SARSA learned the Q function over 3000 episodes using a α of 0.5 and a ϵ of 0.1. The policy was using ϵ-greedy. The TD(0) estimate value function is close to the true value function but notable is lower on the rightmost middle state. The policy is almost optimal, but there are a few more states moving towards state B than for the optimal policy.

A set of 8 learning curves were generated varying α and ϵ between 0.05 and 0.72 with the other value held at the default value listed above. With too low of a learning rate SARSA doesn't learn an optimal policy (likely doesn't discover that moving to state A is better than moving to state B even in the right half of the world). At significantly higher learning rates the learning curve does seem to find the optimal policy, but there is a large amount of variance between episodes. As exploration rates increase the variance in the learning curve increases, and at very high exploration rates SARSA spends so much time exploring that it is not exploiting the known rewards enough.

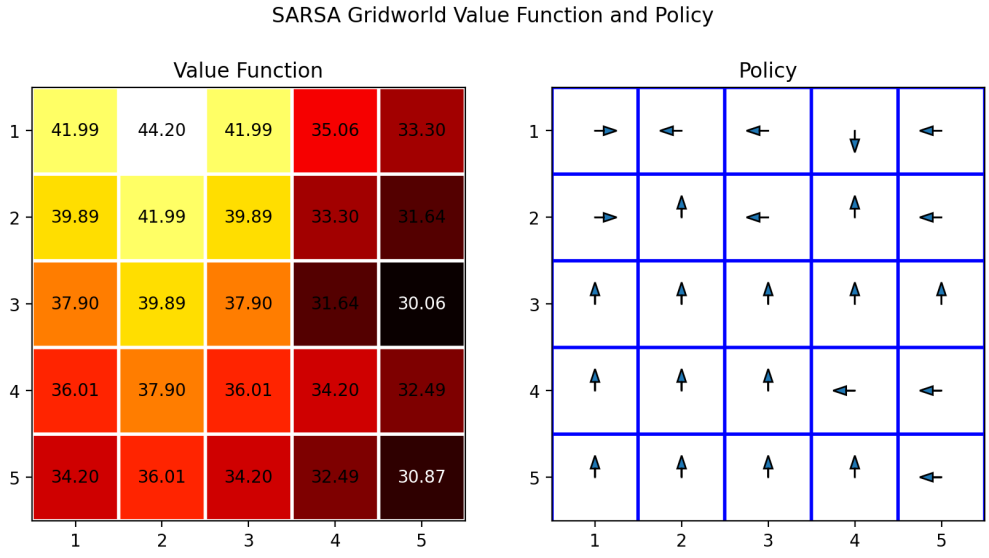

Fig. 8 SARSA gridworld learned value function and policy

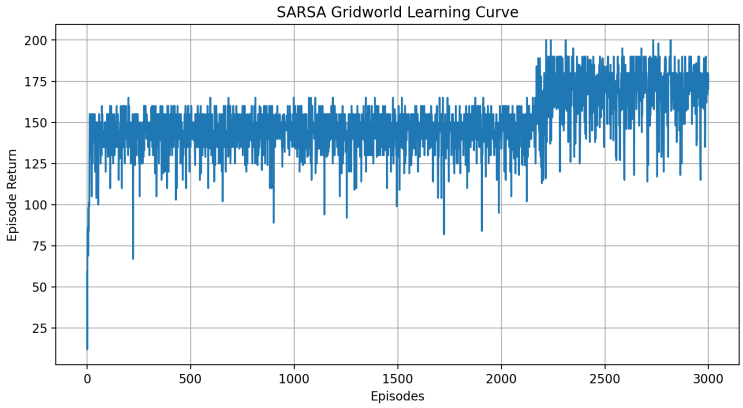

Fig. 9 SARSA gridworld learning curve

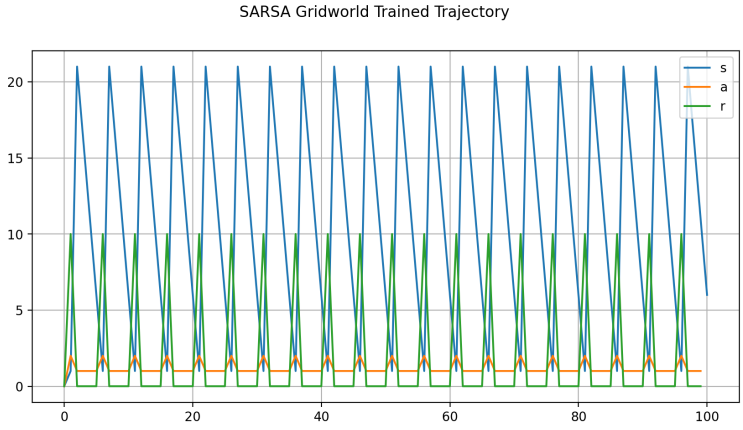

Fig. 10 SARSA gridworld example trajectory

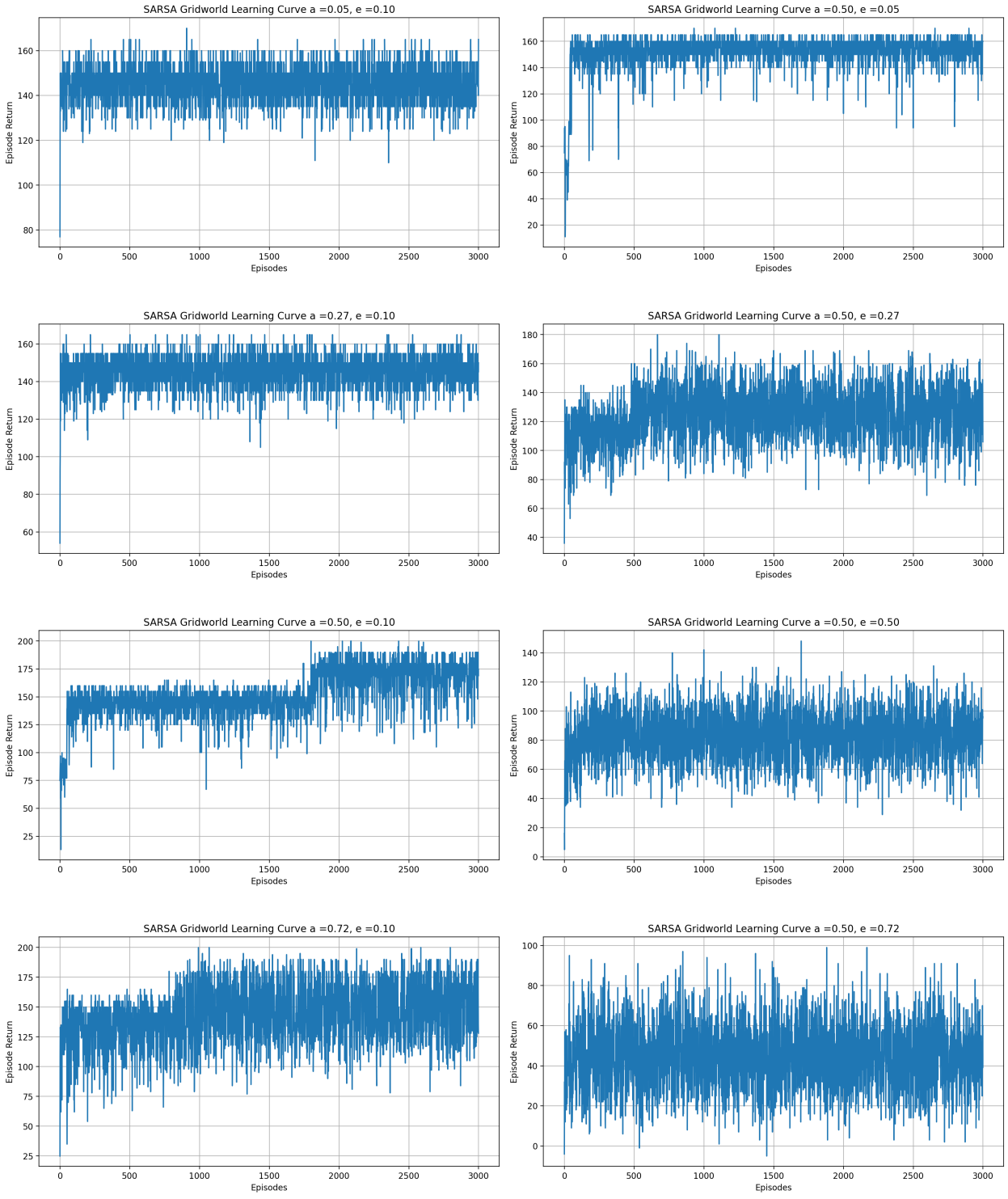

Fig. 11 SARSA gridworld learning curves with varying α and ϵ

4. Q Learning

Q Learning learned the Q function over 3000 episodes using a α of 0.5 and a ϵ of 0.1. The policy was using ϵ-greedy. The TD(0) estimate value function is close to the true value function and significantly closer than SARSA was. The

policy is also almost optimal, but there are a few more states moving towards state B than for the optimal policy. In the learning curve, Q Learning finds the optimal path sooner than SARSA, but has more variance and a very slightly lower average return than SARSA due to its offline policy learning.

A set of 8 learning curves were generated varying α and ϵ between 0.05 and 0.72 with the other value held at the default value listed above. With too low of a learning rate Q Learning doesn't learn an optimal policy (like SARSA). At significantly higher learning rates the learning curve does seem to find the optimal policy, but there is not a large increase of variance between episodes likely because Q Learning learns the optimal policy rather than the safe policy faster. As exploration rates increase the variance in the learning curve increases, and at very high exploration rates Q Learning spends so much time exploring that it is not exploiting the known rewards enough.

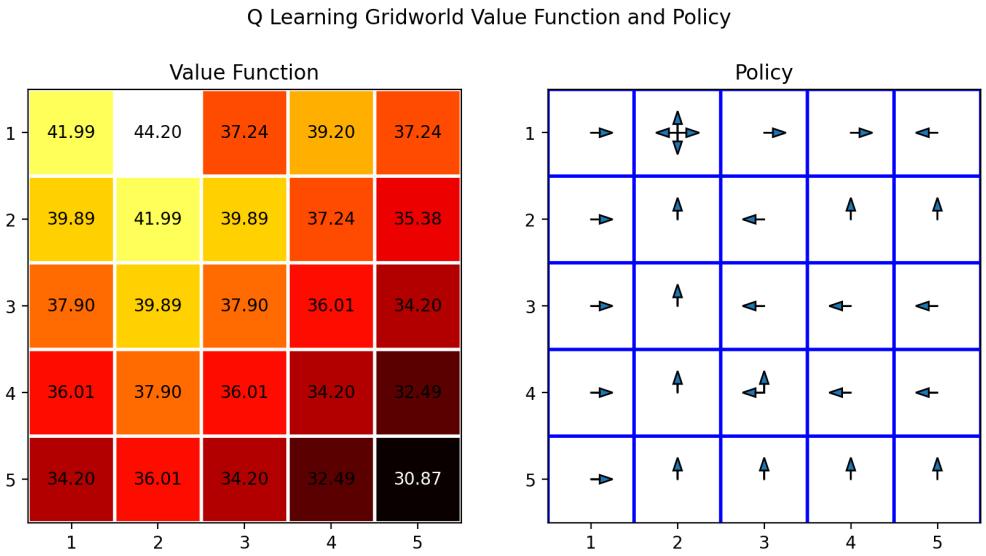

Fig. 12 Q Learning gridworld learned value function and policy

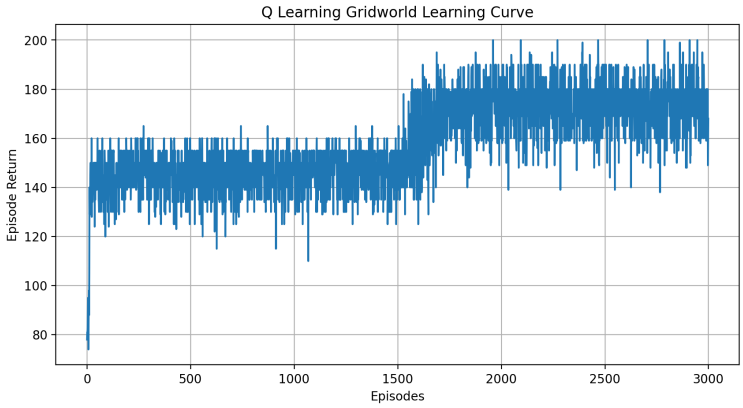

Fig. 13 Q Learning gridworld learning curve

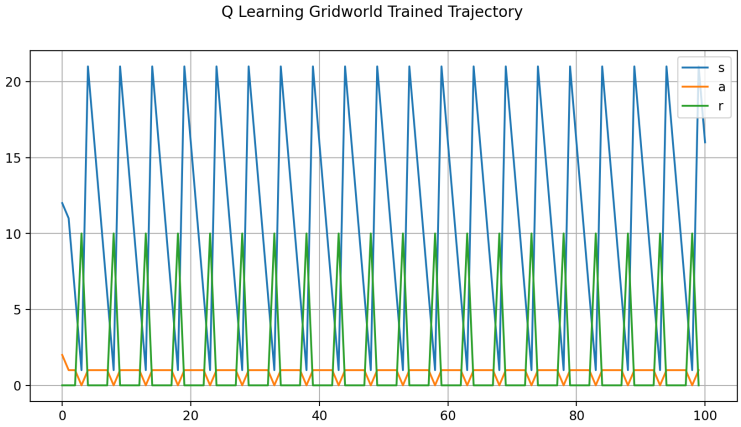

Fig. 14 Q Learning gridworld example trajectory

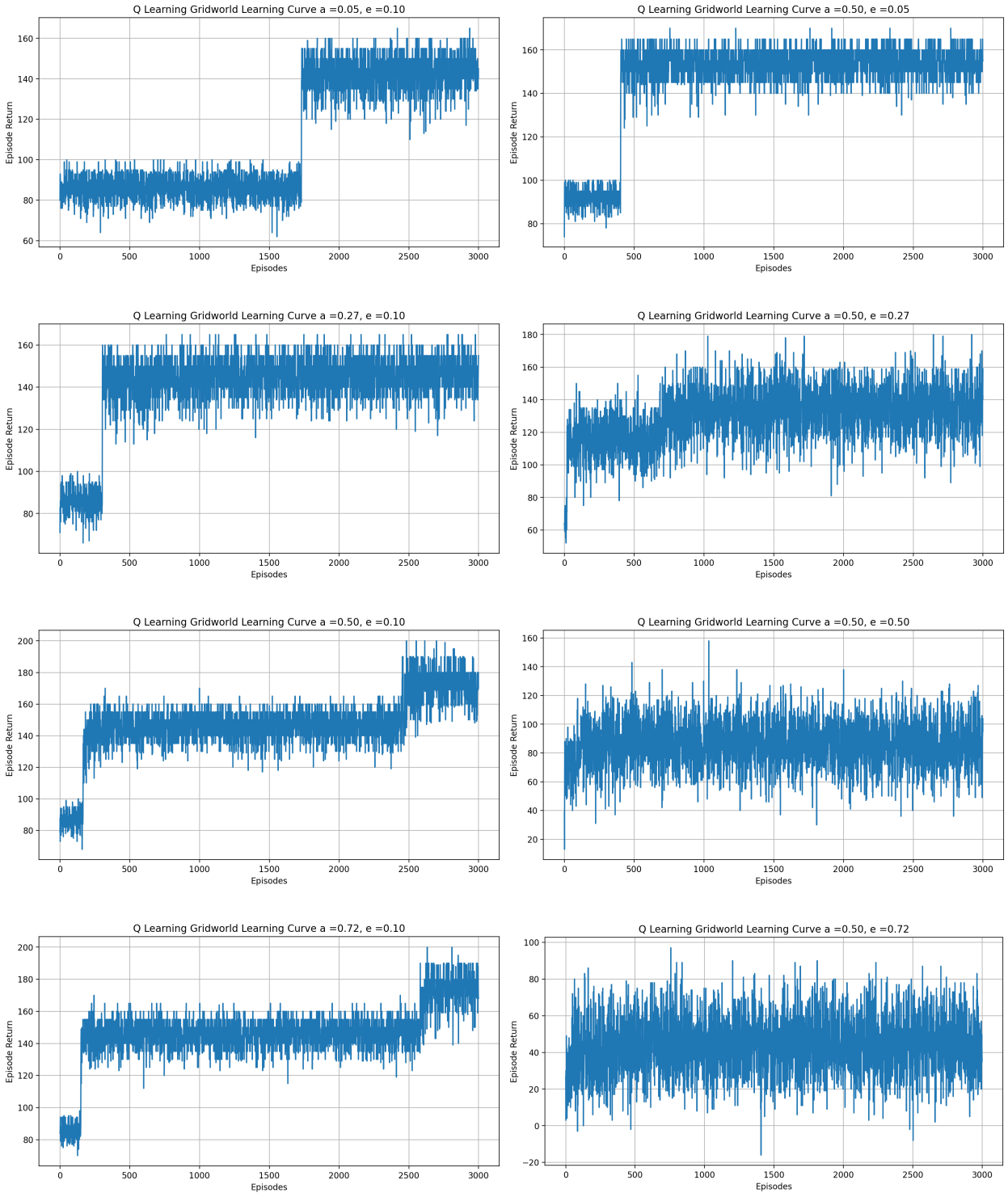

Fig. 15 Q Learning gridworld learning curves with varying α and ϵ

B. Discrete Pendulum

1. SARSA

SARSA learned the Q function over 5000 episodes using a α of 0.25 and a ϵ of 0.1. There are 9765 state-action pairs. The policy was using ϵ-greedy. The TD(0) estimate value function shows that the most left and right states corresponding to the highest absolute angular rates should be avoided due to the very large penalty of going over speed. The center position corresponds to an inverted position with low velocity which would be ideal for maximizing the reward. The policy is not yet optimal after 5000 episodes which can be seen in the example trajectory where the pendulum only stays near the inverted position for short periods. In the learning curve it doesn't look like the algorithm has converged yet which makes sense because with 5000 100 step episodes, there have only been a total of 500,000 state action pairs which corresponds to an average visit rate of 50 per state action pair which will not be equally divided. It seems like the learned policy is to try to balance, but to start constantly rotating if balance can't be achieved to score a point per revolution.

A set of 8 learning curves were generated varying α and ϵ between 0.05 and 0.72 with the other value held at the default value listed above. With too low of a learning rate SARSA doesn't learn how to balance the pendulum at all, but does quickly learn how to avoid the over speed penalty in every combination of exploration and learning rate. The best learning rate appears to be between 0.27 and 0.5 and the best exploration rate appears to be between 0.05 and 0.27. At significantly higher exploration rates the learning curve does not grow much because the exploration is sabotaging the attempts to invert the pendulum.

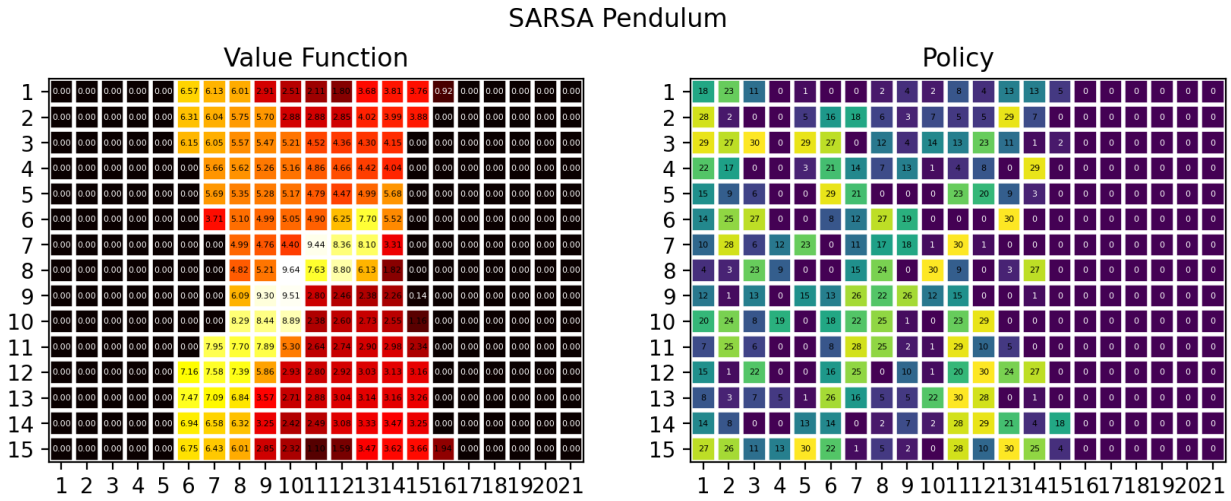

Fig. 16 SARSA pendulum learned value function and policy

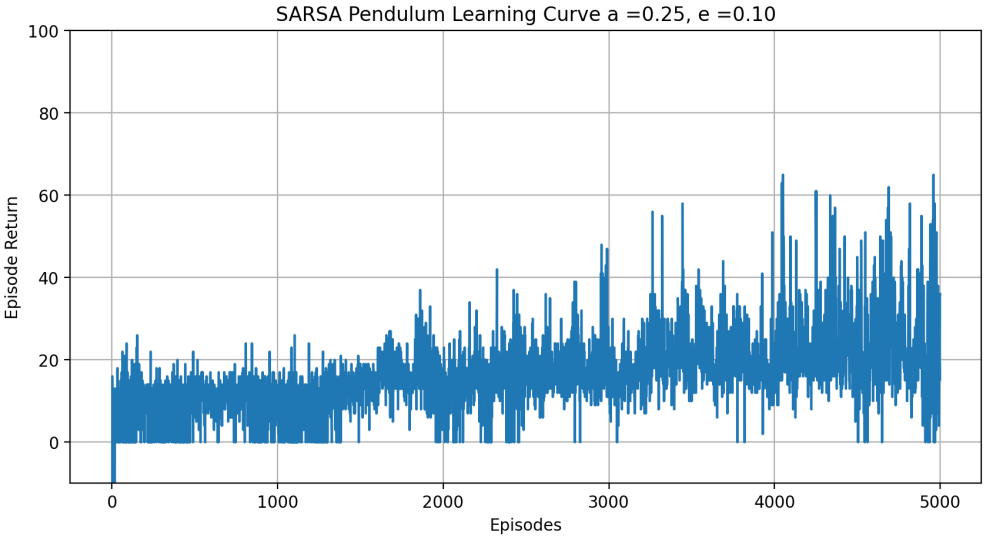

Fig. 17 SARSA pendulum learning curve

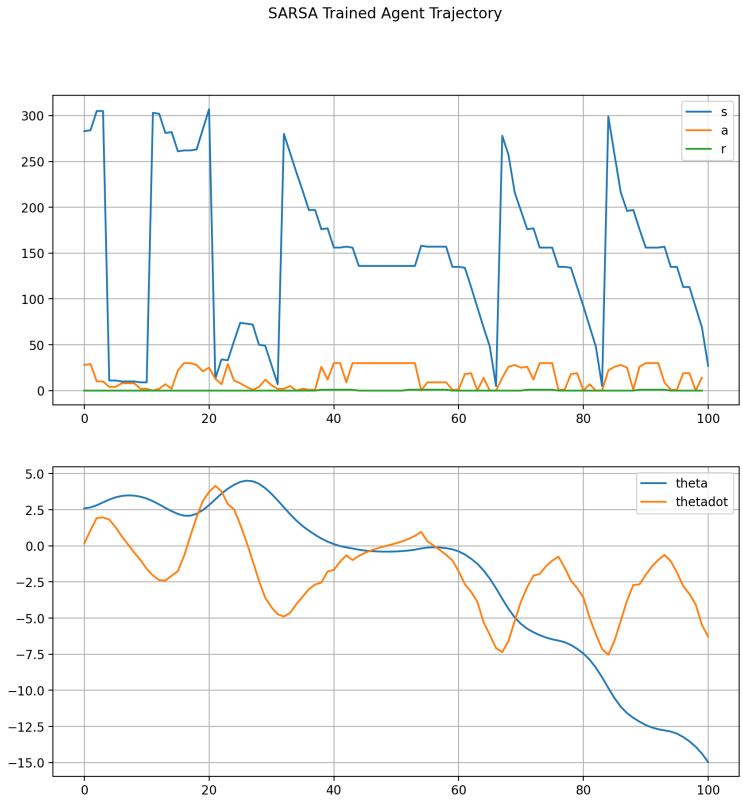

Fig. 18 SARSA pendulum example trajectory

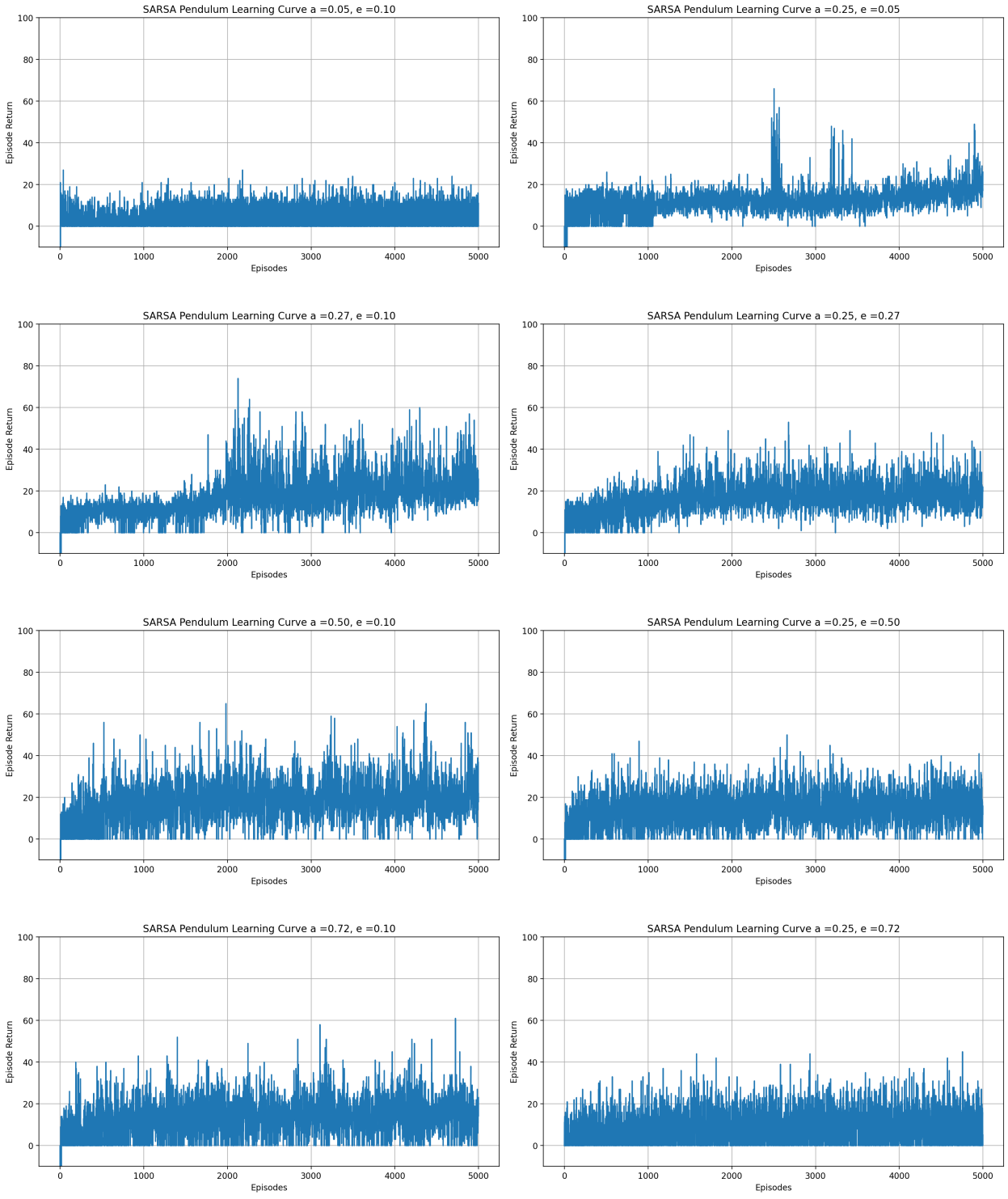

Fig. 19 SARSA pendulum learning curves with varying α and ϵ

2. Q Learning

SARSA learned the Q function over 5000 episodes using a α of 0.25 and a ϵ of 0.1. There are 9765 state-action pairs. The policy was using ϵ-greedy. The TD(0) estimate value function shows that the most left and right states

corresponding to the highest absolute angular rates should be avoided due to the very large penalty of going over speed. The center position corresponds to an inverted position with low velocity which would be ideal for maximizing the reward. The policy is not optimal after 5000 episodes which can be seen in the example trajectory where the pendulum tries to reach inversion early in the episode, but quickly starts rotating to earn a small reward each revolution. In the learning curve it doesn't look like the algorithm has converged yet which makes sense due to the large number of state-action pairs like in SARSA. Q-Learning recieves the speed penalty much more often than SARSA did because Q Learning is much more likely to learn a closer to optimal (but also riskier policy. Interestingly the learned value function shows that Q Learning is willing to move towards high speed states if the pendulum is near the uninverted position (because this is more likely to reach inversion despite the increased penalty risk).

A set of 8 learning curves were generated varying α and ϵ between 0.05 and 0.72 with the other value held at the default value listed above. With too low of a learning rate SARSA doesn't learn how to balance the pendulum at all, but unlike SARSA does not learn how to avoid the over speed penalty in most combinations of exploration and learning rate except for the high exploration rate curves likely due to their inability to pick up significant speed due to the random . The best learning rate appears to be be around 0.5 and the best exploration rate appears to be around 0.27. At significantly higher exploration rates the learning curve does not grow much because the the exploration is sabotaging the attempts to invert the pendulum like with SARSA.

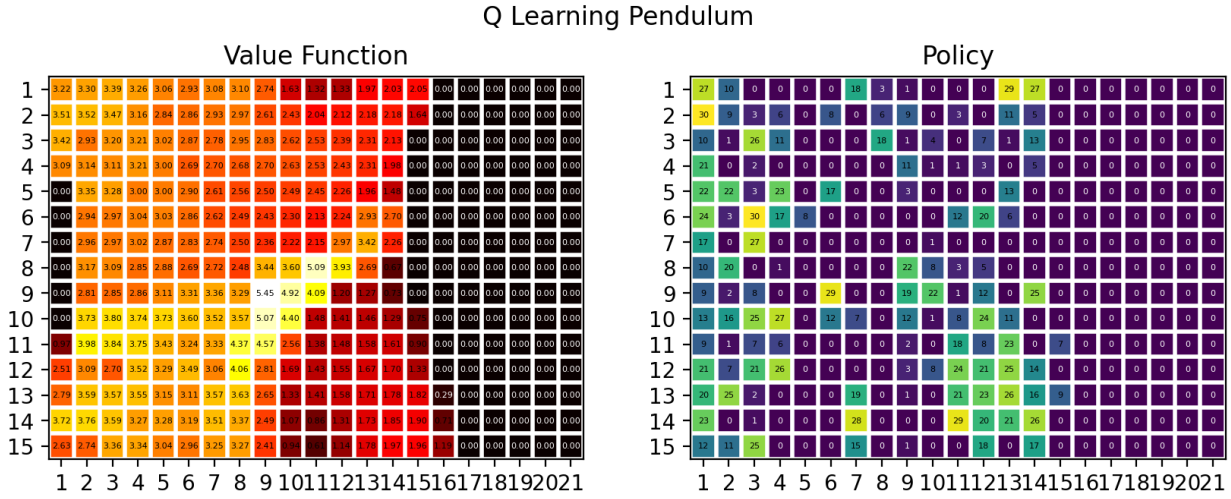

Fig. 20 Q Learning learned value function and policy

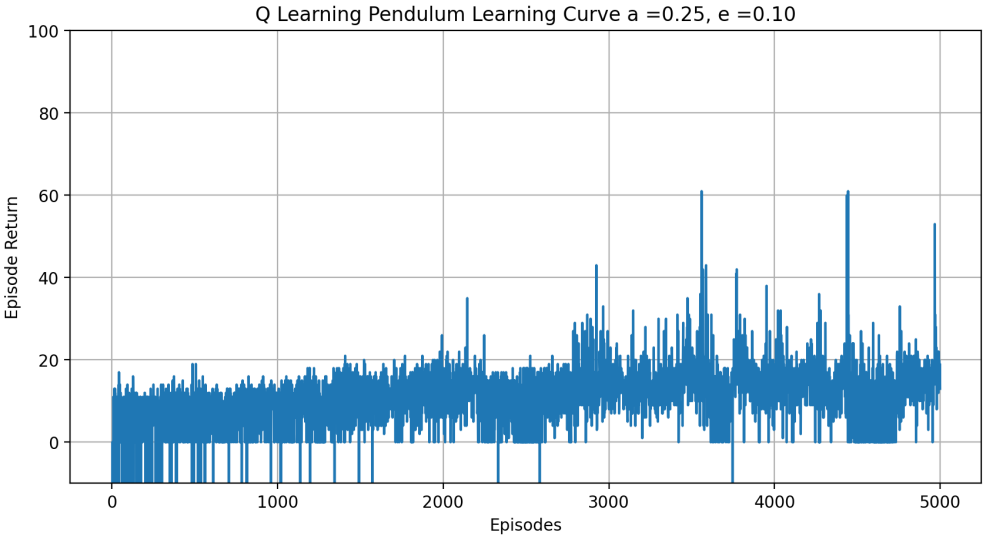

Fig. 21 Q Learning pendulum learning curve

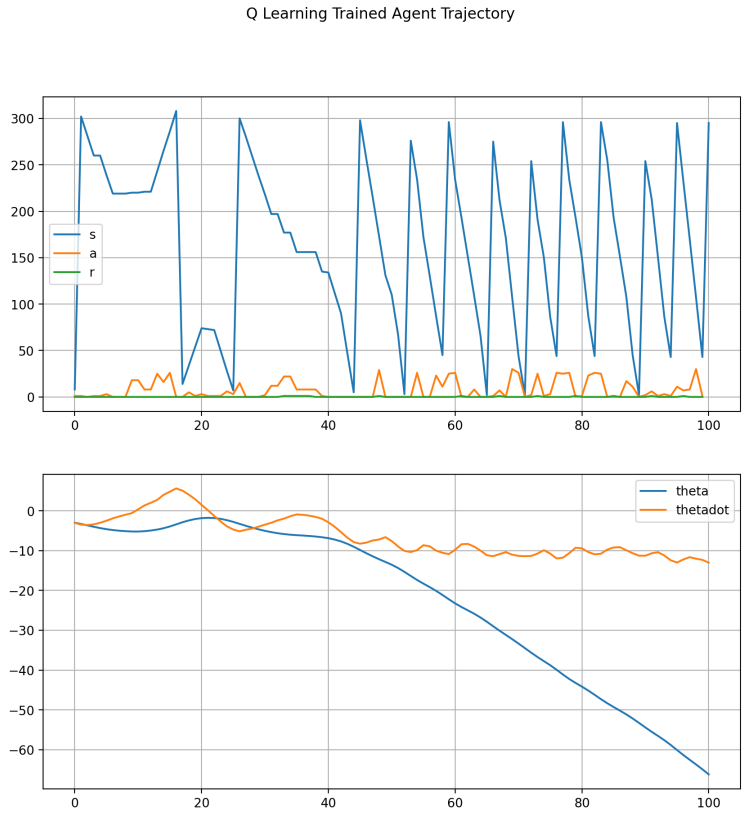

Fig. 22 Q Learning pendulum example trajectory

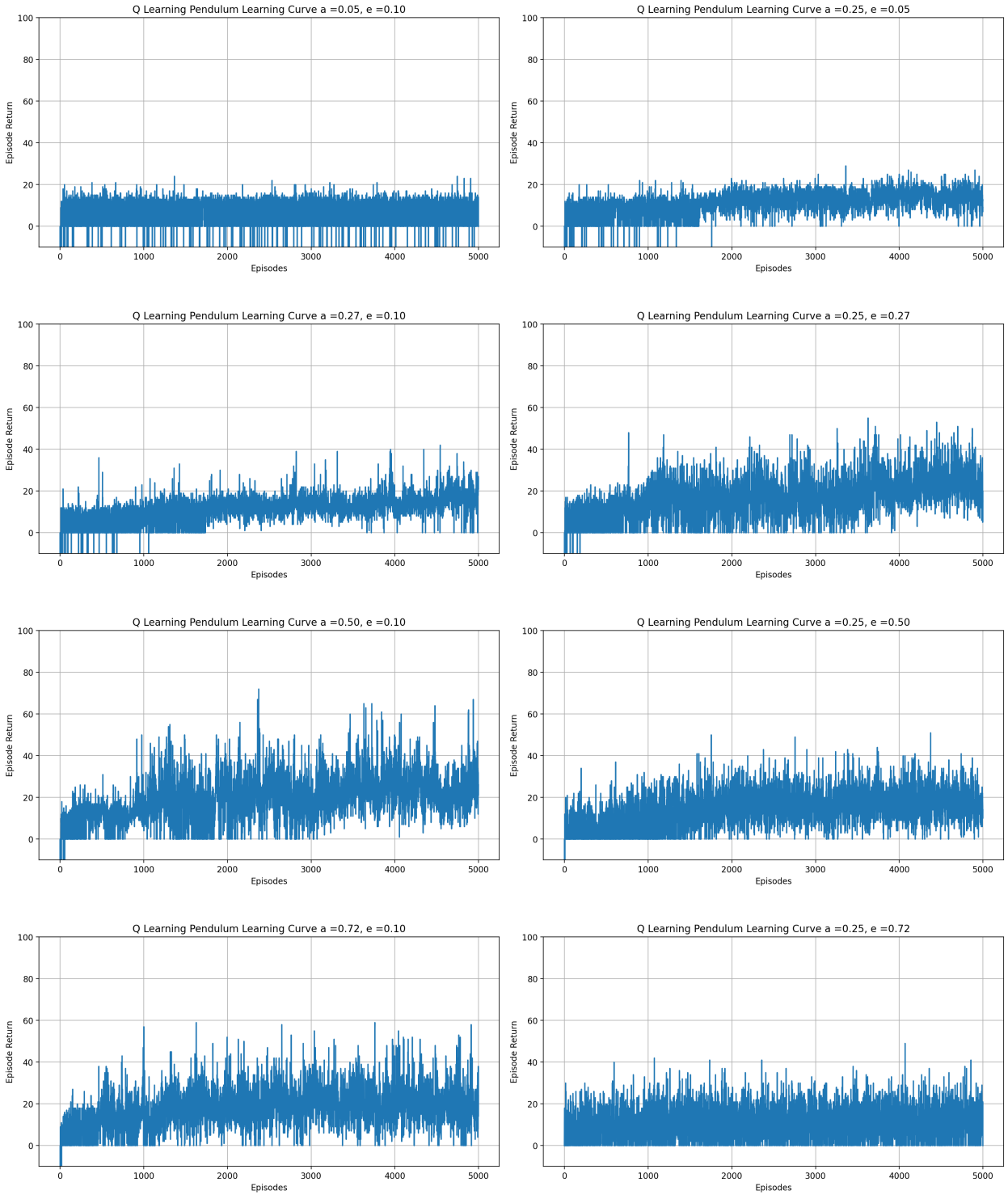

Fig. 23 Q Learning pendulum learning curves with varying α and ϵ