ROB 537 Assignment 3

November 6, 2015

1 N-Armed bandit

1.1 Problem setup

The problem consists of a five-armed bandit each of which reward payout was constructed using a normal distribution with means 1, 1.5, 2, 2, 1.75 and variance 5, 1, 1, 2, 10 respectively. At each timestep, the agent picks an action and gets a reward. The action taken by the agent corresponds to picking the bandit (pulling a lever) and the reward the agent receives is the reward associated with that bandit determined by its mean and variance. The objective in this problem for the agent to maximize its total reward for an episode which is defined to be a given number of arm pulls.

1.2 Relevant Parameters kept constant

- **Learning rate**

  After trying different values for the learning rate, I found a learning rate of 0.1 to be ideal for our setup.

- **Action-Value table initialization = 1.65**

  The action-value was initialized to be 1.65 which is the average of all our bandit's mean payout. Varying this initialization value had some interesting results and will require further analysis. I will vary this parameter and discuss its implications later.

- **Number of Epochs = 250** The number of epochs represent the total learning iterations. Although the cumulative reward for each algorithm are plotted per epoch, this remains a useful parameter in the sense that it should be large enough to show convergence. Running for an arbitrarily large number of epochs can provide a trivial way to satisfy this condition but a large number of epochs will cloud the plots hampering its readability and information density. In these series of experiments, I have found that 250 epochs is generally good to achieve convergence and provides an ideal balance between readability and information volume.

- **Value Update Equation**

  V(a) = (1 - learning rate) * V(a) + learning rate * reward

- **Epsilon (E-greedy algorithm) = 0.1**

  After trying different values for the learning rate, I found a learning rate of 0.1 to be ideal for our setup. This value of epsilon afforded enough exploration for our epsilon-greedy algorithm to converge after 250 epochs which was selected given the reasons above.

1.3 Results

Each algorithm along with its parameters were given 20,000 statistical runs. The relatively high amount of statistical runs was necessitate by the fact that in this problem, our algorithms and the bandits both relied heavily on random number generators. The random generators are not perfect in generating random numbers albeit having a high period. To ameliorate this limitation, I chose a very high number of statistical runs (20,000). Each datapoint and its respective error bars thus are computed using the respective 10,000 datapoints. The final result shows the average of these datapoints and the error bar represent the standard error of the mean (SEM).

1.3.1 Epoch with 10 timesteps

Each epoch consisted of 10 timesteps along which, the reward achieved by picking a bandit were summed to find the total reward. The total reward is then plotted against the epochs in the following plots.

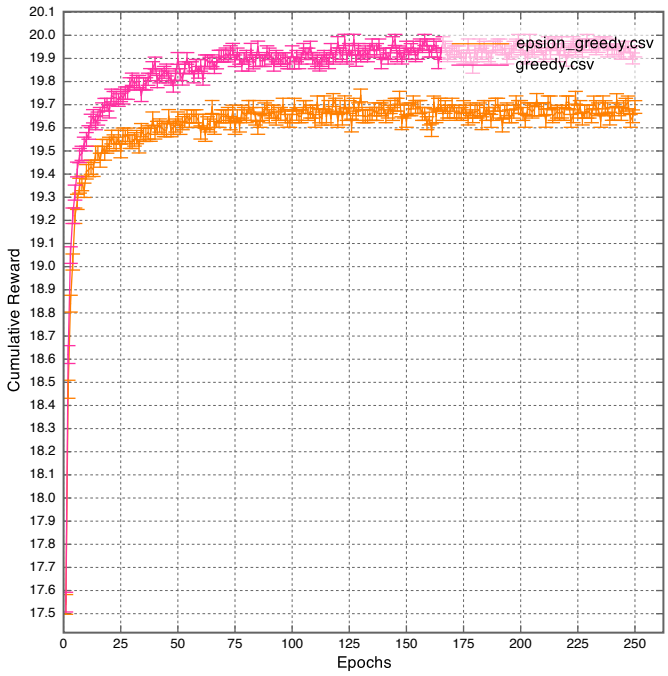

Figure 1: Value initialized to 1.65

The greedy algorithm converged to its optimal value (20) which translates to a mean reward of 2 in approximately 125 epoch while the e-greedy algorithm converged to its optimal value (19.7) which translates to a mean reward of 1.97 in 75 epochs. The greedy algorithm however performed better than the epsilon-greedy algorithm in this experimental setup by a small amount. This is an unexpected result however not a surprising one upon further investigation. An important parameter of note is the value initialization parameter which states the value, the value table is initialized to. This value had a profound effect on the algorithm's performance. Below is a set of graphs obtained modifying the initialization parameter.

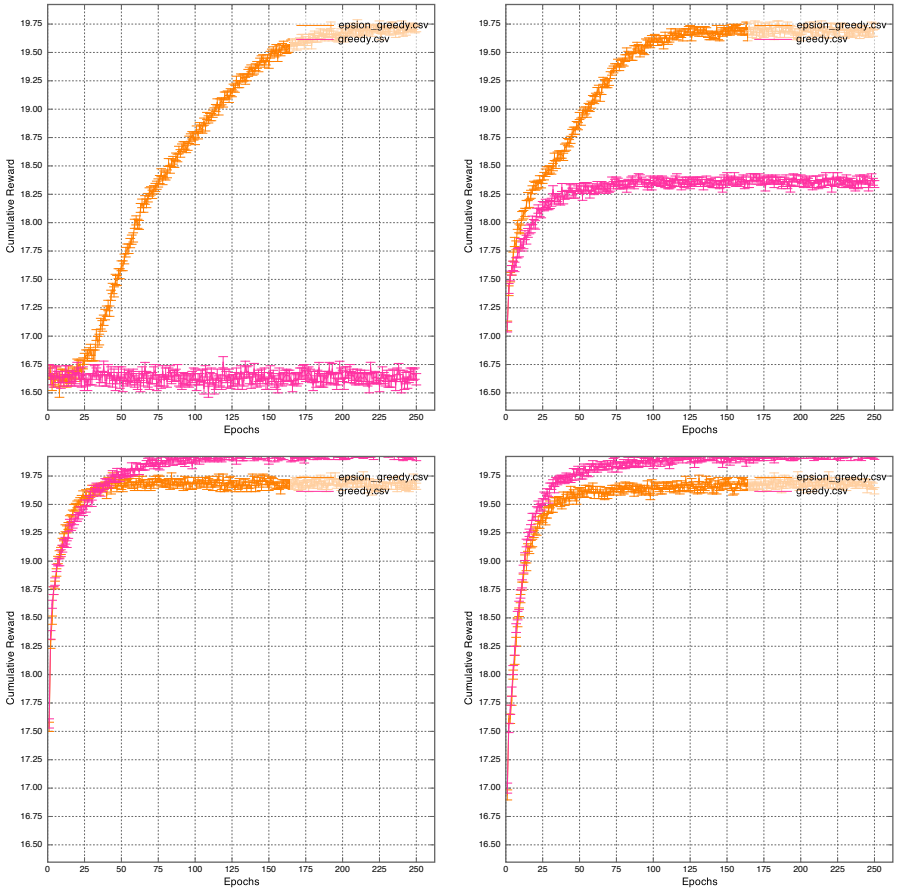

Figure 2: Value-table Initialization values: **-2(left top), 0(right top), 1(bottom left), 3(bottom right)**

The graphs above clarify the reason for the high performance of the greedy algorithm when initialized with a high value for value-table initialization. The trend seem to be that as the initialization value is increased, the greedy algorithm continually improves in its performance even outperforming epsilon-greedy for very optimistic initialization values. An optimistic initialization of the value table seem to favor the greedy algorithm. This is because an optimistic value table inherently causes exploration even for the greedy algorithm. If the value table is optimistic, a reward drawn from one of the bandit is unlikely to increase this already high action-value. This means the greedy algorithm implicitly explores all of its actions before starting to exploit. This implicit exploration is the reason why the greedy algorithm was able to have good performance with optimistic action-value table. Similarly, for a pessimistic action-value table initialization, the greedy algorithm is more likely to stick with the first bandit it picked as its immediate reward is very likely to improve the action-value given already low initialization values. This is why pessimistic initialization causes the greedy algorithm to be perform so bad.

The epsilon-greedy algorithm with an explicitly defined exploration parameter seem to be consistent with the maximum reward it is able to obtain regardless of the value-table initialization. The benefits of an optimistic value-table initialization are however felt in the the rate of learning for this algorithm. As visible in the graphs, an optimistic value seem to accelerate learning and have it converge to the optimal value quickly while a pessimistic initialization caused slower convergence to its optimal value. This is a reasonable result as an epsilon-greedy algorithm for the large part (1-epsilon) is a greedy algorithm and the implicit exploration afforded by an optimistic initialization is still present here too. This implicit exploration benefit in this algorithm however manifests itself in terms of quicker learning.

1.3.2 Epoch with 100 timesteps

The following plots were obtained using 10,000 statistical runs instead of 20,000 as it provided fairly reliable data. Each epoch consisted of 100 timesteps along which, the reward achieved by picking a bandit were summed to find the total reward. The total reward is then plotted against the epochs in the following plots.

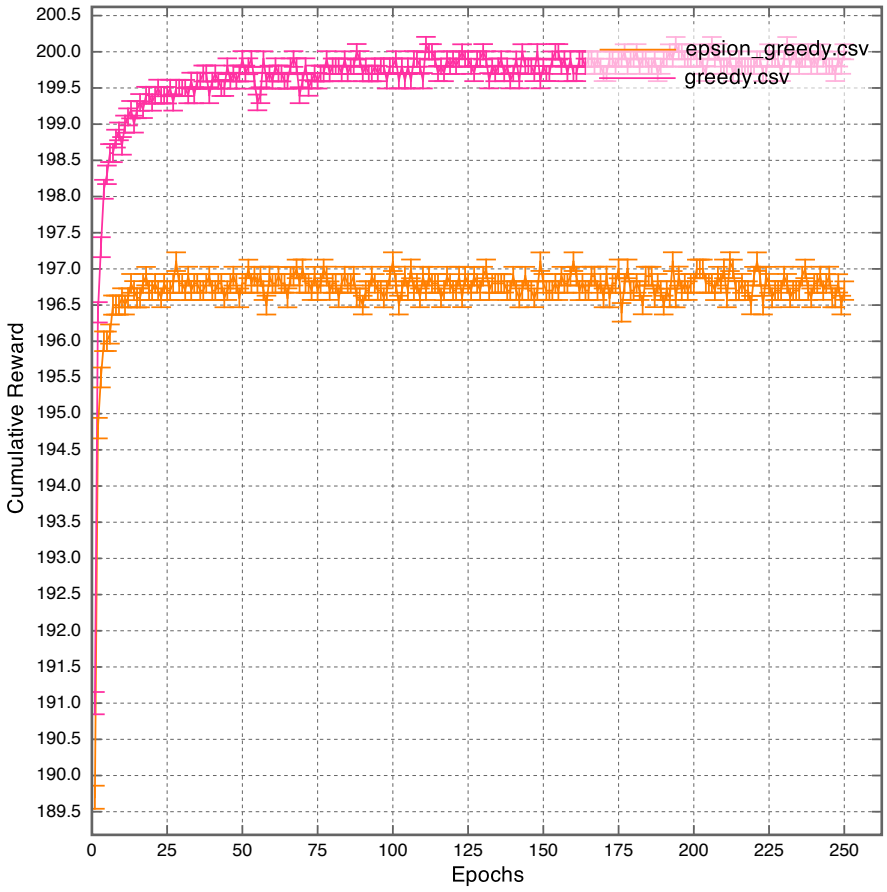

Figure 3: Value initialized to 1.65

The greedy algorithm converged to its optimal value (200) which translates to a mean reward of 2 in approximately 75 epoch while the e-greedy algorithm converged to its optimal value of (197) which translates to a mean reward of 1.97 in 25 epochs. The convergence values are identical to the values achieved in the 10 step N-armed bandit problem. Both learning algorithms converge to the same optimal values respectively but only achieve this quicker in the 100-timestep problem as compared with the 10-time step problem. This is a rather expected result as the 100-timestep problem more chances of picking the bandit and getting feedback because the larger number of steps. With more lever pulls and feedback from the environment, with all other parameters constant we would expect the learning algorithm to converge quicker as is the case.
The greedy algorithm performed better than the epsilon-greedy algorithm in this experimental setup by a small amount. This is an expected result however not a surprising one upon further investigation just like the experiment with 10 timesteps. Below is a set of graphs obtained modifying the value-table initialization parameter.

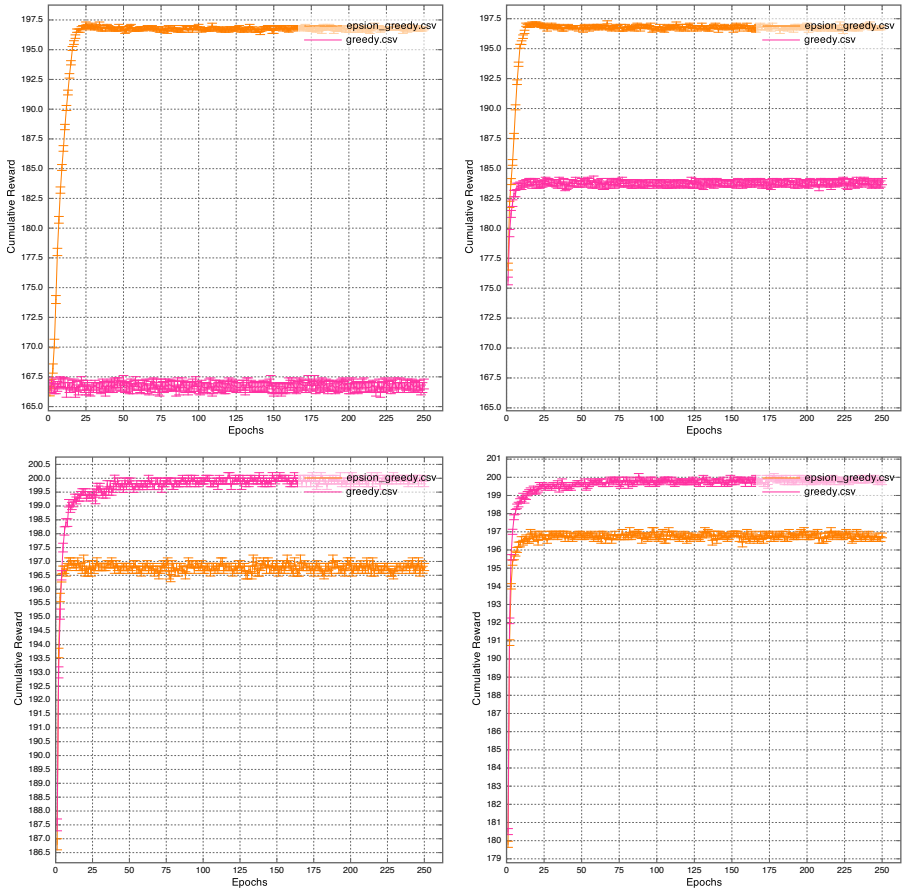

Figure 4: Value-table Initialization values: **-2(left top), 0(right top), 1(bottom left), 3(bottom right)**

Here we see the same trend as with the experiment in the 10 step problem.

The greedy algorithm depends on the value table being initialized optimistically to learn good rewards while the epsilon greedy algorithm merely benefits from it generally in terms of learning speed while not depending on it. A pessimistic value-table initialization in the 100 step domain really hurts the greedy algorithm's performance as shown above. In absence of implicit exploration benefits of an optimistic value-table initialization, the greedy algorithm fails to perform well. An optimistic initialization however due to its implicit co-ordination benefits makes the greedy algorithm slightly better than the epsilon-greedy algorithm. The thing to note however that this improvement is very small and is especially so compared with the drawback of using a greedy algorithm with pessimistic value-table initialization. Also this improvement is due to the lack of exploration in the later stages by the greedy algorithm after an initial implicit exploration created by the optimistic value-table initialization. I hypothesize that the epsilon greedy with decaying epsilon (less exploration at later stages) will achieve similar performance as greedy for optimistic initializations as the constant epsilon (exploration) is the part that bottlenecks this algorithm's performance in later epochs. The following plot demonstrates this assertion. All parameters are kept constant from earlier runs. The action-value table is initialized at 3 (the most optimistic value with the best greedy performance). The main change is that I have implemented a step-wise decaying epsilon such that the epsilon decays to zero after 200 epochs and the epsilon greedy algorithms basically becomes greedy for the last 50 runs.

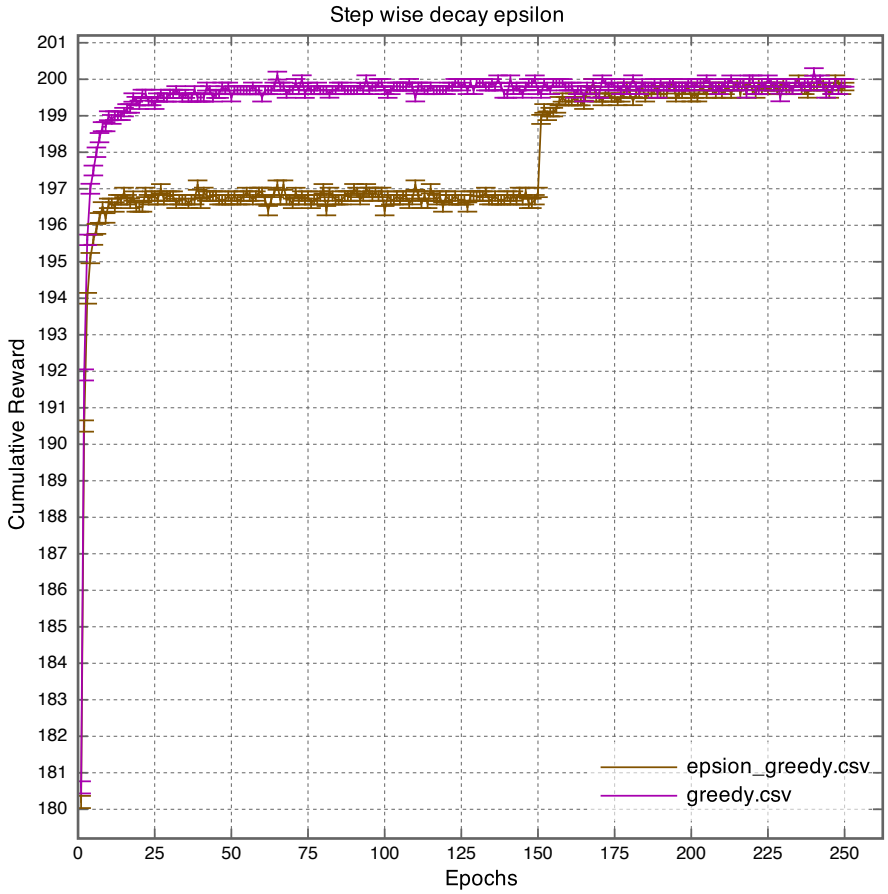

Figure 5: Value-table Initialization values: **-2(left top), 0(right top), 1(bottom left), 3(bottom right)**

1.4 Conclusion

Both algorithms converge to the same respective values in both the 10-step and 100-step problem with the only difference being that the 100-step problem converges quicker than the 10-step one. Also note that the best mean reward performance (2) is in fact the optimal value, one would expect as it represents the highest mean among our bandits (Bandit 3 and 4). Furthermore, while greedy was able to slightly outperform the epsilon greedy algorithm given an optimistic value-table initialization, the improvement was very slight and this was matched when the epsilon greedy was slightly altered to have a decaying epsilon. The difference in performance when started pessimistically was very large between the epsilon-greedy and greedy with epsilon greedy being the clear winner of the two. The The epsilon greedy algorithm is also the more robust of the two algorithms as greedy algorithm depends on the value table being initialized optimistically to learn good rewards while the epsilon greedy algorithm merely benefits from it in terms of learning rate. This means that the epsilon greedy

algorithm is able to converge to good rewards no matter what the initialization value table are while the same cannot be said about greedy algorithms. This makes epsilon greedy the clear better choice among these two algorithms as in most real life situation, the optimism/pessimism of value-table initialization may not be easy to to determine or controlled.

2 Gridworld

2.1 Problem setup

The setup spans a 5X10 gridworld with discrete steps and deterministic transitions between the states. Implementation wise, each co-ordinate in my gridworld represents the grid of which it is the bottom left edge of. For example (0, 0) represents the first positive square block. Using this formulation my 5X10 grid spans 0-9 in the x-axis and 0-4 in the y-axix all inclusive. The goal location is set to be at (9, 1) representing the last x-block and the second from bottom y-block. At the start of each statistical run the agent is spawned at a random location that is not the goal state within the grid world. There are five actions available to the agent at any time and each action causes a deterministic transition in state. If the agent goes past the artificial boundary set in this problem the agent is put back in the same state as its last allowed one and it is assessed the same penalty/reward as the its last/(now current) state. In other words, there is no penalty for falling off the edges.

2.2 Reward setup

The agent is assessed a penalty of -1 each timestep that it is not in the goal state. If the agent reaches the goal-step, there is no exit parameter and the agent keeps taking actions. In simple words, the simulation **does not end** when the agent reaches the goal. The simulation only ends at 20 timesteps no matter what the agent does. The agent can get multiple rewards from being in the goal state (reward of 100) every time it is in that grid. This setup was chosen to increase the magnitude of gradient information of getting to the goal quicker. In this setup, getting to the goal quicker has a significant value as the agent can in optimal case go to goal and stay there. If the simulation was ended after reaching to the goal, the difference of an agent reaching to goal in n steps and n + 1 steps starting from the same position will only be 1. This is very small difference and the experimental setup I chose aims to increasing this gradient information thus insentivizing agents to get to the goal zone as quick as possible and also learn to stay there.

2.3 Metric design

- **Total Reward**

  The total reward accumulated by the agent over twenty time steps were recorded and plotted against the epoch.

- **Percent Optimal Reward**

  The total reward accumulated is an obvious and easily implementable metric, however it has severe limitations and can be mis-representative of

agent performance. The cumulative reward accumulated by an agent for example depends heavily on the location within the gridworld, the agent was initialized. If an agent is initialized in a grid very close to the goal state, the total reward accumulated may be serendipitously high without much real credit to the learning algorithm. To ameliorate this limitation I introduce "Percent Optimal Reward" a metric that captures the percent of optimal reward that the agent received. The percent optimal reward is calculated using the following equation:

percent optimal reward = (100 * (total Reward + total time step) /(optimal reward + total time step))

The optimal reward is calculated at the start of each epoch when the initialization location of the agent is determined randomly. The optimal reward is calculated as:

optimal reward = 100 * (total time step - optimal steps + 1) + (optimal steps - 1) * (-1) where the optimal steps is the least number of steps between the goal co-ordinate and initialization co-ordinate of the agent.

The Percent Optimal reward in a way normalizes the performance of the agent with respect to the optimal possible performance given the initialization condition. Note that the cumulative reward of an agent can range from -20 to a maximum of 2,000 given different starting conditions. The Percent optimal reward dis-associates the random input of starting position and gives a level metric such that both an agent initialized at co-ordinate (0, 0) i.e very far away from goal and an agent initialized at co-ordinate (9, 0) i.e one step away from goal, have equal opportunity to achieve the same performance optimality of 100 percent if they follow the optimal policy.

2.4 Action-Value Learner

The action value learner has a action-value table with five actions available. This is the same action-value learner used in solving the N-Armed bandit. The values of each action is updated after each time-step according to the reward from the environment.

V(a) = (1 - learning rate) * V(a) + learning rate * reward

The parameters setup is very similar to the N-armed bandit and is mostly kept constant. The value-initialization table was also kept constant at 0 as we are only using the epsilon-greedy algorithm which we determined from Experiment number 1 to be robust to the pessimitic/optimistic action-value initialization. The number of epochs was lengthened to observe the learning progress for a longer time while the learning parameters was varied. 1000 statistical runs were done to obtain the plots below.

- **Action-value initialization = 0**
- **Epsilon = 0.1**
- **Number of Epochs = 1000**: Given more epochs to settle

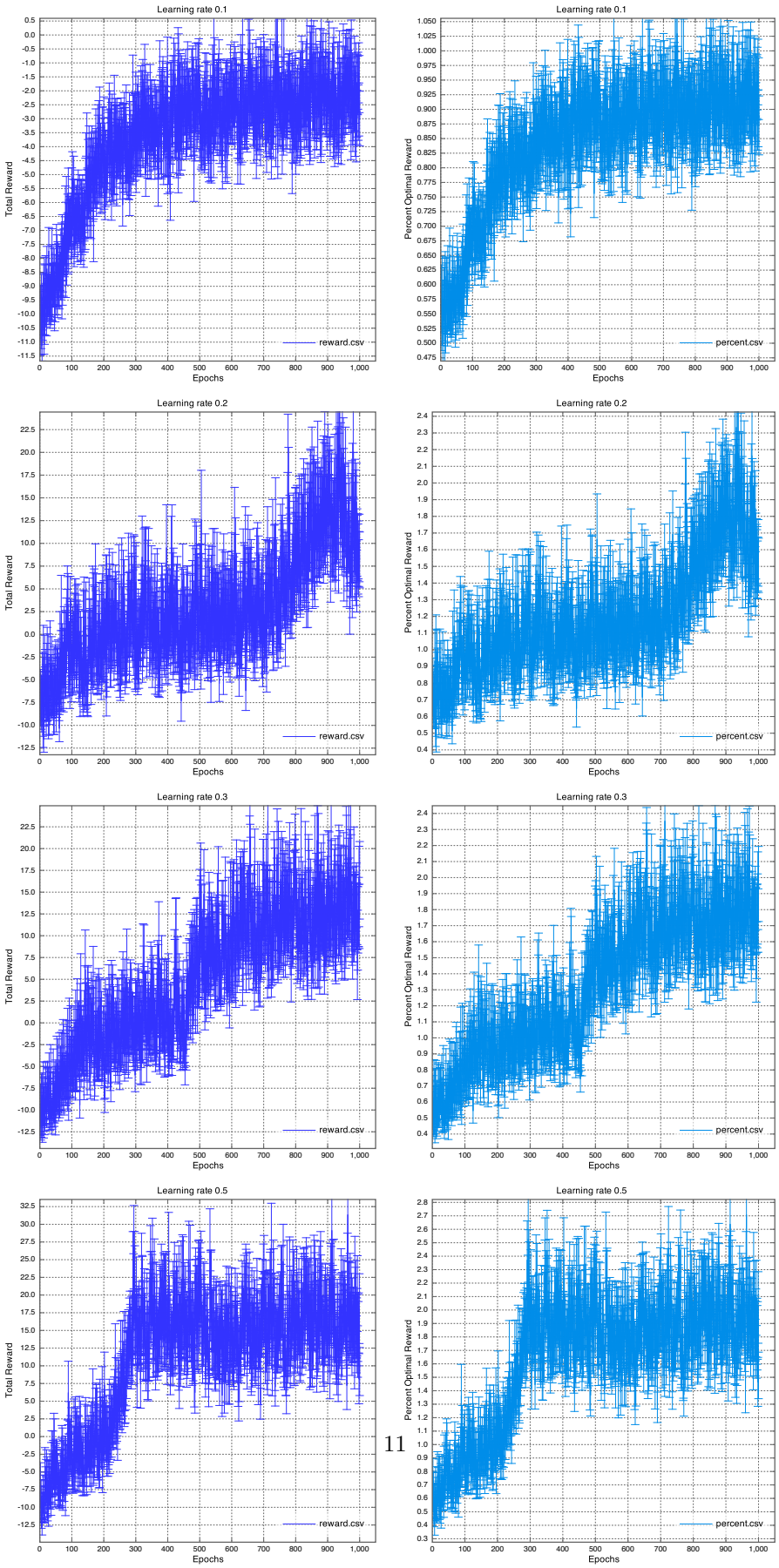

Figure 6: Action-value learner

The action-value learner failed to learn good policies to reach to goal as shown by the results above. The best Percent Optimal Reward it was able to achieve was approximately 2.5 percent with a learning parameter of 0.5. This is a terrible performance and shows the ineffectiveness of action-value learner to learn good policies in stateful domains such as this. This is a rather expected result as the action-value learner neither considers state nor the availability of future rewards while picking an action. This causes it to be hardly better than what a random walker could achieve in this domain. The following plot is obtained using a random walker(no value function), keeping all parameters the same from the above experiment.

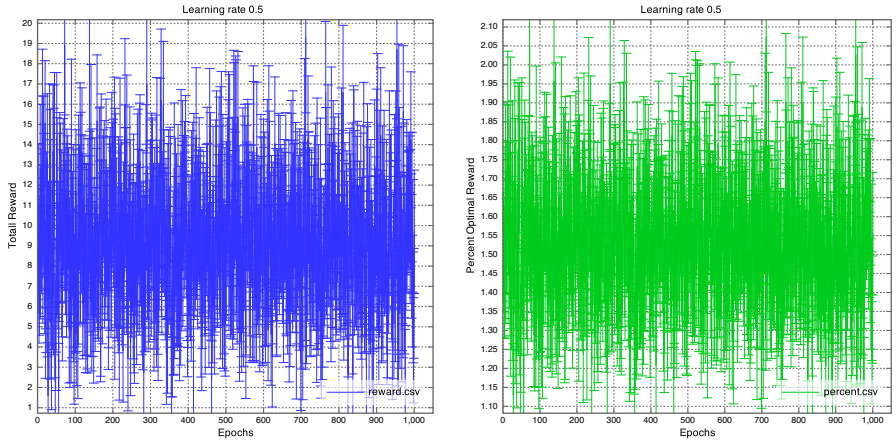

Figure 7: Random walker

The random walker achieves a Percent optimal performance of approximately 1.6 percent (mean) with high uncertainty which is not much statistically different than the action-value learners 2.5 percent. This perhaps highlights the action-value learners inability to learn good policies in stateful domains with delayed reward. Although every state transition (action) had a reward associated with it, the goal reward (100) was only available at a certain arbitrary timestep. The agent depending on its initialization value could take 4 of 5 actions available to it at some timestep to experience this goal reward. This means the positive reward signal if the agent achieved the goal was used to update the value of the action value that achieved that final transition to the goal state or stayed there instead of the path the agent took to get to the goal state. This is a major reason why this action-value learner value failed to learn good policies (which translates into path in this problem) to achieve optimal reward. In sum, the lack of states and delayed reward which caused incorrect assignment of credit cause the action-value learner to fail so miserably in achieving good policies in this domain.

2.5 Q-Learner

The Q-learner has a Q-table which builds upon the action-value table by adding state information onto it. In simple words, it takes the action-value table of the previous experiment and extends it to each possible state (co-ordinate in this

setup). This means that we have 50 separate action-value tables that corresponds to and only to each state (co-ordinate) in the 5X10 grid. The Q-values of each action is updated after each time-step according to the reward from the environment as described by the following equation.

$$Q(s_t, a_t) = (1 - \alpha_t)Q(s_t, a_t) + \alpha_t \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a)\right] \tag{1}$$

where,
$\alpha_t = learningrate$
$\gamma = discountfactor$
$max_a Q(s_{t+1}, a)$ = the maximum Q-value from the next state

At every timestep, the agent takes an action that depends on its policy (epsilon greedy and its variant). The policy chooses this action based on the Q-values of the action corresponding to the current state. After taking the action, the agent receives a reward $R(s_t, a_t, s_{t+1})$ from the environment. The agent then calculates the maximum Q-value of the next state (the one it transitions with the action chosen) and uses this and the reward to update the Q-value of its current state (before transition) using the above equation.

The parameters setup is very similar to the action-value learner except the addition of γ (discount factor). The discount factor can be thought of as a way to consider delayed rewards so that the state/action pairs that lead to good reward will also be rewarded (partially) with the reward that they lead to. The policy used was a step-wise epsilon greedy as tried out in the last N-armed bandit experiment where we follow epsilon-greedy for the first 80 percent of epochs, at which point the epsilon decays to 0 and the epsilon greedy basically turns into a greedy algorithm.

The value-initialization table was kept constant at 0 as we are only using the epsilon-greedy algorithm which we determined from Experiment number 1 to be robust to the pessimitic/optimistic action-value initialization. The number of epochs was lengthened to 1500 to observe the learning progress for a longer time while the discount factor was varied. 1000 statistical runs were done to obtain the plots below.

- **Action-value initialization = 0**
- **Number of Epochs = 1500**: Given more epochs to settle
- **Learning factor = 0.1**:
- **Step-wise decay epsilon-greedy with epsilon decay at 0.8 total Epoch**:
- **Epsilon = 0.1**

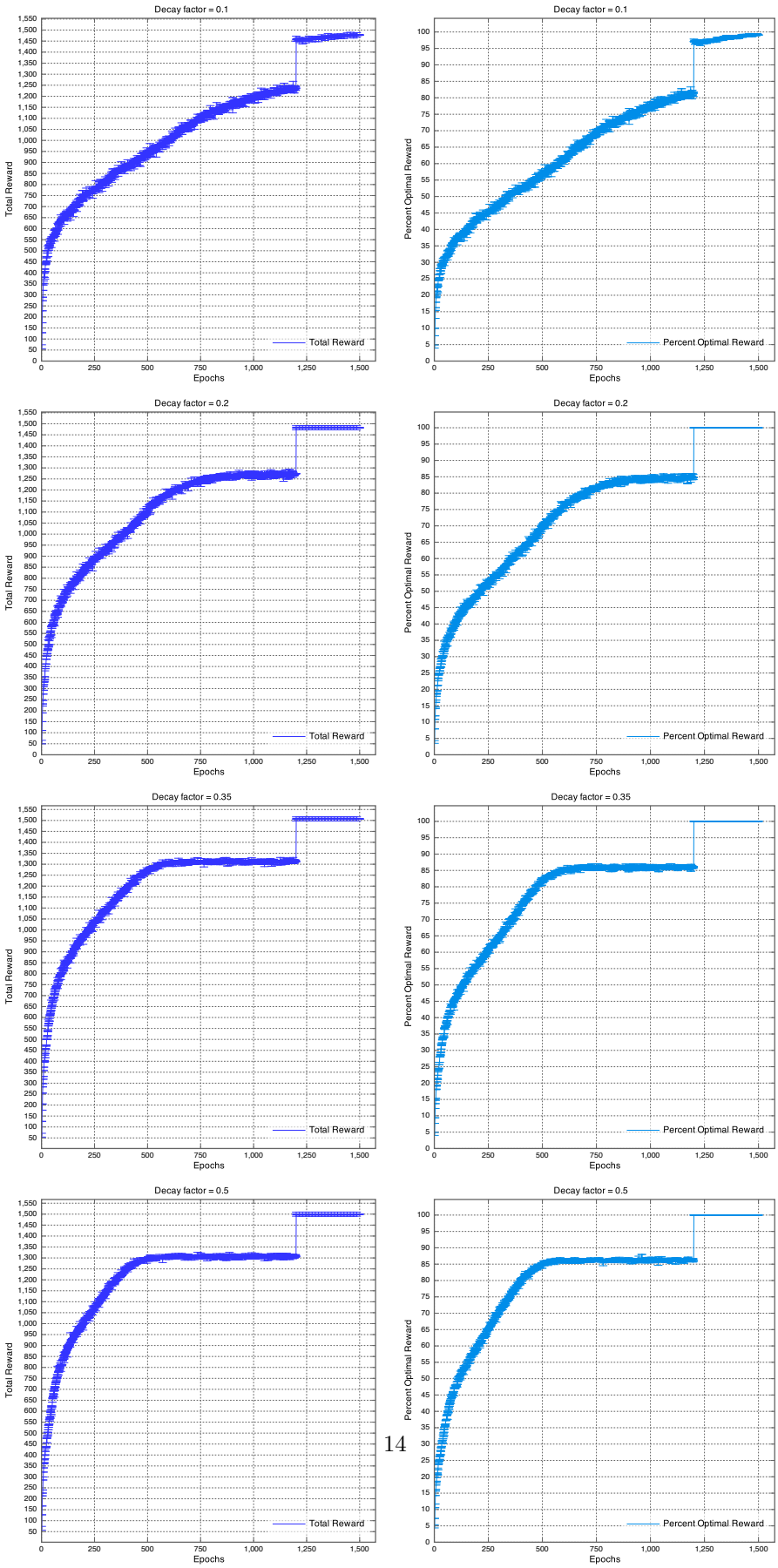

Figure 8: Q-learner with varying discount factors

The Q-learner seem to do a very good job of learning good policies of reaching to goal and staying there. In fact, when the epsilon greedy algorithm decays into a greedy algorithm after, 1200 epochs (80 percent total) we can see the Q-learner performing close and at optimal levels. this is signified by the graphs to the right that show the percent optimal rewards of close to and at 100 percent. Recall that our percent optimal functions are agnostic to the serendipidity of random initialization position, which makes this achievement of optimal policy that much more meaningful and relevant. This means that, irrespective of starting position, our agent was able find the optimal policy signified by 100 percent Percent Optimal reward.

The discount factor seem to effect the learning rate of our agent. A higher discount factor of 0.5 was observed to have the quickest convergence while lower discount factors seem to converge more slowly. With a discount factor of 0.5 the step wise epsilon greedy algorithm converged within 500 epochs while the lowest value discount factor achieved this in approximately 1500 epochs. While this observation may suggest a higher discount factor to be a clear better choice, it must be noted that a high discount factor is not always ideal for all domains. Our domain was deterministic and an action led to the same state change each time with no uncertainty. This determinism was an important factor in our observation regarding the discount factor. A stochastic domain with non-deterministic state transition might not fare too well with high discount factors.

Furthermore, As evident from the plots, the epsilon greedy algorithm is able to converge very close to the optimal policy. It converges to a value of 85 percent percent optimal reward all four cases. This is an expected result as with an epsilon greedy algorithm with a steady epsilon, the exploration would never stop. In simpler words, the algorithm would continue exploring (picking random actions) even after knowing the optimal solution due to the steady epsilon value. Our epsilon value being 0.1 which translates to 10 percent exploration would lead to the agent picking random actions 10 percent of the time despite perhaps already having learned the optimal path. This ties in nicely with our observation.

The step-wise epsilon greedy function as discussed in the earlier N-bandit problem causes this steady explicit exploration to decay into no exploration (changing epsilon to 0) after epoch 1200. this leads to the agent following the optimal path (using the Q-values) without any exploration whatsoever. In other words, after using some explicit exploration to find the optimal path, it switches to full exploitation once it has learned that path. This leads to it converging to the optimal policy and achieving 100 percent optimal reward. Burrowing from our observation from the N-armed bandit problem I wanted to try how a greedy algorithm with a high initialization parameter would fare in this domain. All parameters are kept constant above and the discount factor of 0.5 is selected while the Q-values are initialized optimistically to 10. The number of epochs were also extended to 2000.

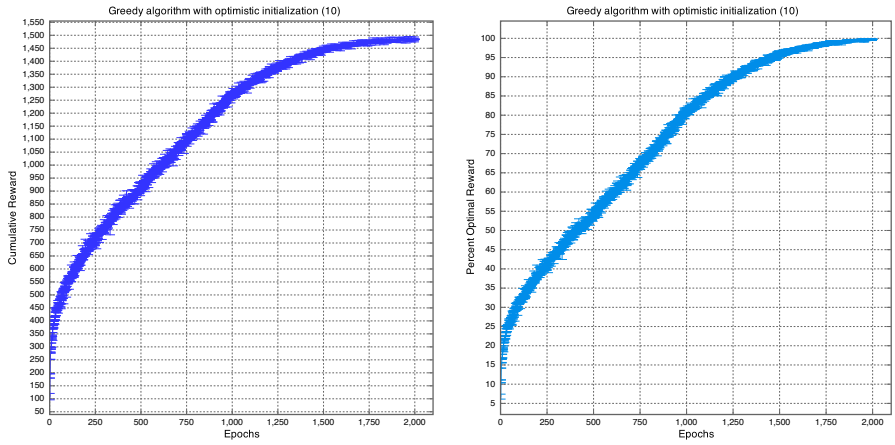

Figure 9: Greedy Algorithm with optimistic initialization

As expected, the greedy algorithm with optimistic initialization does find the optimal path but it is very slow (2000 epochs) to converge. This is due to its dependence on implicit exploration rather than an explicit one to search its solution space that leads to slow but steady convergence.

2.6 Conclusion

The action-value learner failed to learn good policy for the 20 step gridworld problem. The Q-learning agent on the other hand was able to find the optimal policy to this 20-step gridworld problem with the highest learning speed to convergence of 500 epochs. The reason for this disparity in performance between the two learning algorithms can be traced to the primary and key difference between them:

- **State information**
  The Q-learning algorithm integrates state information in its Q-table in form of state-action pairs while the action-value learner does not have state information. This is a fundamental flaw in the action-value learner in terms of dealing with stateful domains like the gridworld. The lack of state information means that the action-value learner only associates the reward/penalty (feedback form the environment) with the action it took to get that feedback. The state involved along with that action responsible for that feedback is lost causing fundamental learning problems. For example, assuming our goal state at (9, 1) if our agent is at (9, 0), an go up action would lead to a feedback of goal reward (100). The Q-learner associates this with being in state (9,1) and taking action"up" while the action-value learner attributes this feedback only to the action "up". This means that in the next step, the action-value learner would (with high probability according to epsilon) choose the "up" action as it associated that action with the positive goal reward. This would lead it to exit the goal state and head upward from then on which would not be an ideal policy. The Q-learner however would can then learn a good action "stay" at the goal state independently of what actions it took to get there. This

is the main reason why Q-learning was able to learn optimal policy while the action-value learner failed miserably.

- **Discount factor**:
  The Q-learner integrated a measure of future rewards from a state-action pair in its Q-table apart from the immediate reward it received. This led it to value state-action pairs on their ability not only to get good feedback immediately from the environment but also to maximize future rewards. This was especially important in this gridworld domain as the immediate feedback (-1) in most cases are not very informative. The true reward that the agent seeks (goal reward of 100) is in most cases (initialization) delayed and is only received by the agent after a number of state-action pairs. Thus in order for the agent to learn a good path to goal, it has to be able to attribute good Q-values to the state-action pairs that lead to the goal, not just the immediate state-action pairs that lead directly to goal. Q-learning achieved this by using the discount factor to attribute some measure of current state-action value with the future value of state-action pairs led from the current one. Q-learning also considered the maximum possible future action-values meaning it was optimistic about the its ability to pick the optimal action in the future. The action-value learner in the other hand had no discounting and only updated its values using immediate rewards which was a big factor in its bad performance.

The action-value learner is a good algorithm choice for a stateless problem like the N-armed bandit in Problem 1. When a domain however has state information relevant to it, the action-value learner fails to learn optimal policies as evidenced and explained in Problem 2. In these stateful states, Q-learner due to its integration of state information and delayed rewards, performs much better and is able to learn optimal policies as evidence by the 20-step gridworld problem in problem 3.