RL Programming Assignment 1

Ishu Dharmendra Garg: CS13B06

February 2017

1 Code organization

Following is the code organization:

- Class Dist: This abstract class defines random variable for stationary probability distribution for an arm of the bandit. Since in the given assignment we only need normal distribution for arm, there is only one concrete child class defined called class Normal_Dist which defines a normal distribution with the mean and std passed to it as argument.
- Class Multiarm_Bandit: This class defines a multiarm bandit with each arm having its own separate probability distribution.
- Class Policy: This abstract class defining the general behaviour of how a policy selects arms from a multi_arm Bandit. There are several concrete child classes like eps_greedy, UCB, SoftMax defining how these specific policies selects arm and updates estimate value function. Policy class has various methods like:
  - select: implemented by concrete class defines how to select an arm.
  - update: implemented by concrete class defines how to update the state of policy.
  - reset: Resets the sate of the policy.
  - simulate(T): Runs the policy for T time steps.
  - generate_testbed(n, P, T, m, s): Generates P instances of Multiarm bandits with n arms that with each arm having normal probability distribution with std = 1.0 and mean drawn from a normal distribution drawn from a normal distribution of mean m and standard deviation s.
  - evaluate(n, P, T, m, s): evaluates the policy by evaluating the policy after generating testbed with arguments passed to it. It returns two lists, first is the average reward at different time instances and other is percentage optimal selection at different time instances.

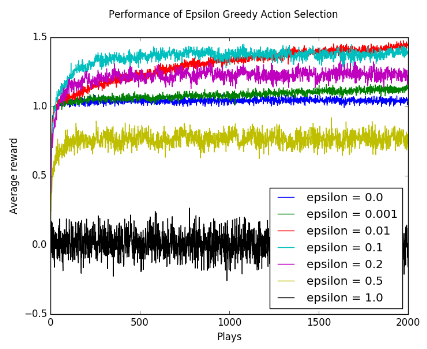
2 About the Code

The code uses runs different instances of policy on different core during the evaluation of policy hence utilizing different cores of the cpu. Also the code is written such that each arm of the multiarm bandit can have different probability distribution. It can also be thought of as a general purpose library (for stationary prob distribution) for implementing and evaluating different policies, where for a policy you just have to define the selection and the update rules.

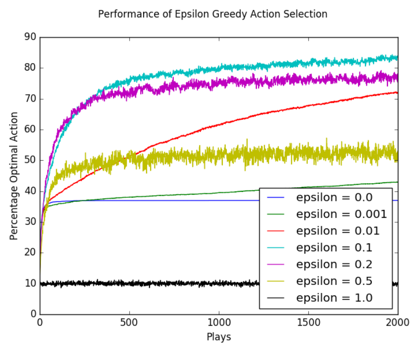
3 Experiments for 10 arms

For the 10 arm bandit problems the test bed were generated with mean = 0, std = 1, P = 5000, T = 2000 and n = 10.

3.1 ϵ Greedy Method



(a) Average Reward for different epsilon



(b) Percentage optimal selection for different epsilon

Figure 1: Performance for Epsilon Greedy Action Selection Policy

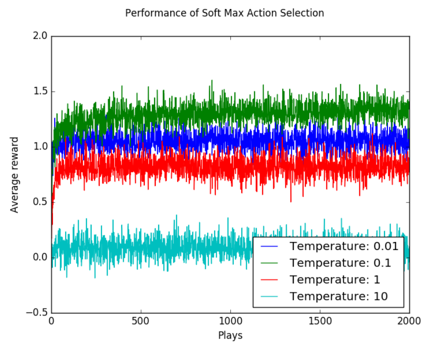
Observations: Point to be noted: Epsilon determines exploration. More eps, more is exploration.

- For epsilon = 1, we can see that the average reward is zero and average optimal selection is 10 which clearly indicates a random behaviour as expected.
- For very less epsilon = 0 and epsilon = 0.001, we can see that till T = 2000 plays do not perform well. This is because they have no or very less exploration. Hence the poor performance. Due to high exploration they have a much steep curve during the beginning but then soon saturates because of lack of exploration.
- For curve with eps = 0.001, we can still see that slow rising of green line. This is because as the time proceeds the value function becomes more

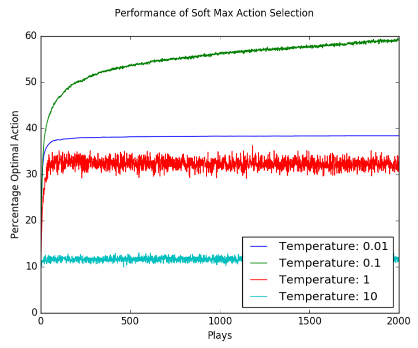
accurate. And since epsilon is smallest positive this policy will have least exploration and hence best performance at the time of convergence (Time >> T = 2000).

- The curve for eps = 0.5 has already saturated and is not expected to rise as we can see that the curve for average optimal selection has already reached 50 %.
- The average reward curve for eps = 0.01 surpasses curve with eps = 0.1 after a some time, which is the expected behaviour.
- For the given time frame Epsilon Greedy Policy with eps = 0.1 performs the best. But if the time period had been longer we may select a policy with lower eps value.

3.2 Soft Max



(a) Average Reward for different temperature



(b) Percentage optimal selection for different temperature

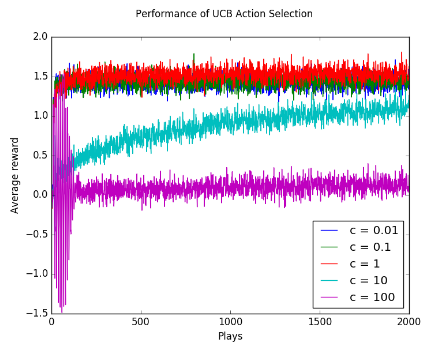
Figure 2: Performance for Soft Max Action Selection Policy

Observations: Point to be noted: Temperature determines exploration. More tem, more is exploration.

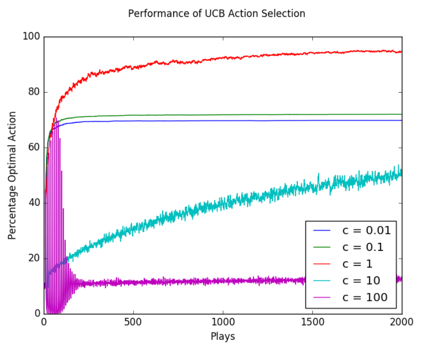
- For very high temperature, tem = 10 we see that the policy behaves randomly (average percentage optimal selection). Also the average reward is close to zero.
- For tem = 1, we see some improvement but still the temperature is too high leading to over exploration.
- For tem = 0.01 we see that it is better than the last two parameters. But we still have problem of over exploitation as seen from the average optimal selection curve. Saturation of this curve before optimal indicates that we are now not exploring enough.

- For tem = 0.1 is the best temperature parameter among all. This finds the optimal balance between exploration and exploitation. The reason can be because tem is of the same magnitude of the expected rewards of the best arm.

3.3 UCB



(a) Average Reward for different c parameter



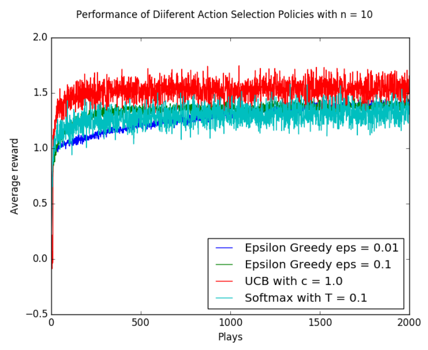
(b) Percentage optimal selection for different c parameter

Figure 3: Performance for UCB Action Selection Policy

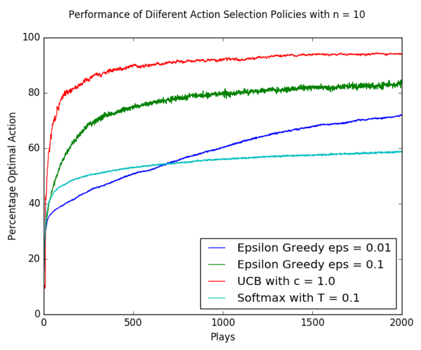
Observations: Point to be noted: c determines exploration. More c, more is exploration.

- For very large value of c= 100, the exploration is the highest and the action selection is almost random as shown by the action selection curve.
- For large high value of c = 10, we still have more exploration but it is steadily decreasing as we can see that the curve for optimal action selection is rising.
- For small values of c = 0.1 and 0.01, we see that rise very fast and sturate very early. This is because of less exploration and more exploration.
- The best parameter is c = 1.0. We see that the average reward is highest and also the optimal action selection curve is rising meaning there is optimal balace between exploration and exploitation.

3.4 Comparison Between Policies



(a) Average Reward for different Policies



(b) Percentage optimal selection for different Policies

Figure 4: Performance for Different Action Selection Policy

Observations:

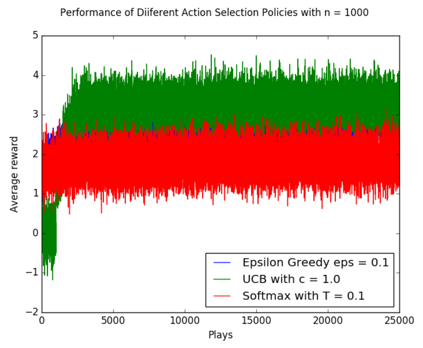
- Softmax performs the worst. This is because the diffence between the best value function and second best action value is not much. For example consider just two arm bandit problem. Let the value function of arm 1 be 2v and that of arm 2 be v where v = 0.1. Even after convergence the probability of selecting arm two is significant. $P = \frac{e^{v/0.1}}{e^{2v/0.1}+e^{v/0.1}} = \frac{1}{e^{10v}+1} = \frac{1}{e+1}$. So the main problem with this method is that even at convergence, it does *significant* unnecessary exploration.
- The performance of Epsilon greedy is decent but not the best. This behaviour can be attributed to never ending exploration. Though the exploration can be arbitrarily reduced but we still explore at convergence. Setting epsilon close to zero would mean very slow convergence and setting it high will mean lesser average reward at convergence. Ideally we would like the epsilon to reduce.
- This policy behaves the best among all the other policies. We can clearly see that it counters the problems of both the methods very efficiently. The best part of the policy is that adjusts the explorations according to the time. Initially exploration is moderate, and then it rapidly it decreases (logarithmically). Also it also takes care the sampling frequency of each of the arm. If the frequency is too less, it is given more preference and vice versa.

4 Experiments for 1000 arms

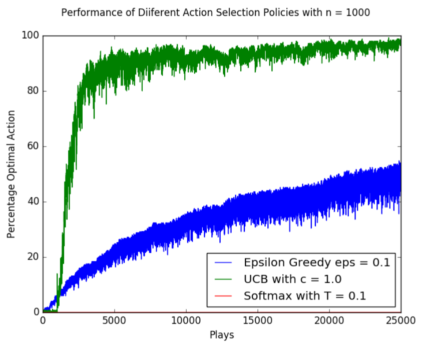
For the 1000 arm bandit problems the test bed were generated with mean = 0, std = 1, P = 200, T = 25000 and n = 1000.

With the limited ram and computation power of my machine, I was not able to rune the experiment for very large time and for many experiments. Due to very high abstract and generalized code, the programs requires a lot of memory and time to rum. I have figured out a few optimizations (specially in soft max policy implementation) but did not implement it in this version code.

4.1 Comparison Between Policies



(a) Average Reward for different Policies



(b) Percentage optimal selection for different Policies

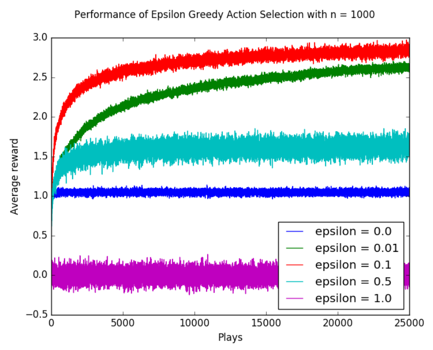
Figure 5: Performance for Different Action Selection Policy

Observations:

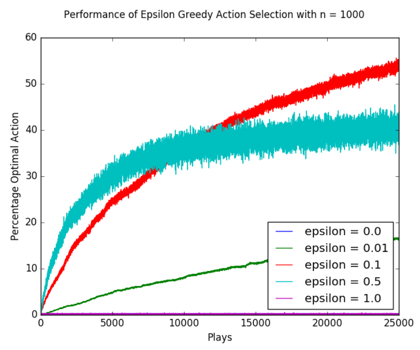
- The sof-max and UCB are not the best method (given the time frame). The time frame is too small and I expect the UCB to perform better in later run (shown by the average optimal selection curve corresponding to UCB curve but for softmax is almost zero and not increasing).
- We can see that Epsilon-Greedy clearly beats other methods. It attains good rewards soon and the average action selection curve also increases rapidly to hight value.Seeing this reward I decided to further explore nature of epsilon-greedy further.

4.2 ϵ Greedy Method

For this experiments the average has been done for 2000 experiments.



(a) Average Reward for different epsilon



(b) Percentage optimal selection for different epsilon

Figure 6: Performance for Epsilon Greedy Action Selection Policy

Observation: The trends in results are very similar to the multi-arm bandit with 10 arms.