CMPSCI 687 Homework 5

Due November 25, 2018, 11:55pm Eastern Time

Instructions: This homework assignment consists of a written portion and a programming portion. Collaboration is not allowed on any part of this assignment. Submissions must be typed (hand written and scanned submissions will not be accepted). You must use LaTeX. The assignment should be submitted on Moodle as a .zip (.gz, .tar.gz, etc.) file containing your answers in a .pdf file and a folder with your source code. Include with your source code instructions for how to run your code. You may not use any reinforcement learning or machine learning specific libraries in your code (you may use libraries like C++ Eigen and numpy though). If you are unsure whether you can use a library, ask on Piazza. If you submit by December 11, you will not lose any credit. The automated system will not accept assignments after 11:55pm on December 11.

- Solution 1:

$$\begin{aligned}(Action)a &\sim N(\theta^t\phi(S_t),\sigma^2)\\ &= \frac{1}{\sqrt{2\pi}\sigma}e^{\frac{(a-\phi(S_t)\theta^T)^2}{2\sigma^2}}\\ \frac{\partial\log(\pi(s,a,\theta))}{\partial\theta} &= \frac{\partial}{\partial\theta}\log(\frac{1}{\sigma\sqrt{(2\pi)}})+\frac{\partial}{\partial\theta}\frac{(a-\theta^T\phi(S_t))^2}{2\sigma^2}\\ &= \frac{(a-\theta^T\phi(S_t))\phi(S_t)}{\sigma^2}\end{aligned}$$

$$\frac{\partial}{\partial\sigma}\log(\pi(s,a,\theta)) = \frac{\partial}{\partial\sigma}\log(\frac{1}{\sigma\sqrt{(2\pi)}})+\frac{\partial}{\partial\sigma}\frac{(a-\theta^T\phi(S_t))^2}{2\sigma^2} = \frac{-1}{\sigma}+\frac{(a-\theta^T\phi(S_t))^2}{\sigma^3} \tag{1}$$

- Solution 2:
- The fixed-point of the TD update using linear function approximation occurs when the expected TD update is zero

$$\begin{gathered} td\_update = (R_t+\gamma\phi(S_{t+1})w^T-\phi(S_t))(\phi(S_t)-\gamma\phi(S_{t+1}))\\ E[(R_t+\gamma\phi(S_{t+1})w^T-\phi(S_t))(\phi(S_t)-\gamma\phi(S_{t+1}))]=0\\ \text{Expanding this we get:}\\ E[R_t(\phi(S_t)-\gamma\phi(S_{t+1})] = w^TE[\phi(S_t)\phi(S_t)^T-\gamma\phi(S_t)\phi(S_{t+1})^T-\gamma\phi(S_t)^T\phi(S_{t+1})+\gamma^2\phi(S_{t+1})\phi(S_{t+1})^T]\\ b=E[R_t(\phi(S_t)-\gamma\phi(S_{t+1})]\\ A=E[(\phi(S_t)-\gamma\phi(S_{t+1})(\phi(S_T)-\gamma\phi(S_{t+1}))^T]\end{gathered} \tag{2}$$

- LSTD algorithm for $TD(0)$
  Given: A batch of data

```
Output: A coefficient vector $w$ for which $V\pi(S_t) \sim w\phi(S_t)$
Set $A := 0, b := 0, t := 0$

For each tuple $(s, a, r, s')$ do: {
        a. Set $A := A + (\phi(s) - \gamma\phi(s'))(\phi(s) - \gamma\phi(s'))^T$
        b. Set $b := b + r(\phi(s) - \gamma\phi(s'))$
}
Invert A and compute $w = (A)^{-1}b$ when required.
```

- LSTD λ algorithm in A Boyan's paper use *e_traces* and expects a simulator to generate episodes. It is not possible to use the following algorithm (LSTD-λ) if our batch of data contains (s, a, r, s') tuples from multiple episodes and if it is not possible to segregate episodes and simulate episodes from this batch of data. In the following algorithm I am assuming we can simulate episodes from given batch of data.
- LSTD algorithm for $TD(\lambda)$
  Given: A batch of data consisting of many (s, a, r, s') tuples. No step sizes necessary.

```
Output: A coefficient vector $\beta$ for which $V\pi(S_t) \sim \beta\phi(S_t)$
Set $A := 0, b := 0, t := 0$

for $n := 0, 1, 2...do:$ {
Choose a start state S$_t \in S$
Set $z_t := \phi(S_t)$
while $S_t \neq S_\infty$ do: {
a. Simulate one step of the chain, producing a reward $R_t$ and next state $S_{t+1}$. Assuming it is possible to simulate an episode using the batch of data.
        Let $s = S_t, a = A_t, r = R_t, s' = S_{t+1}$
        b. Set $A := A + z_t(\phi(s) - \gamma\phi(s'))^T$
        c. Set $b := b + r * z_t$
        d. Set $z_{t+1} = z_t + (\phi(s) - \gamma\phi(s'))$
        e. Set $t := t + 1$
}
}
```

Whenever updated coefficients are desired, invert A and compute $w = (A)^{-1}b$.

- Solution 3:
- Softmax Sarsa Lamda Gridworld Hyperparameters

- $lr = 0.1$ $sigma = 5$ $lambda = 0.2$
- $lr = 0.0316$ $sigma = 10$ $lambda = 0.2$

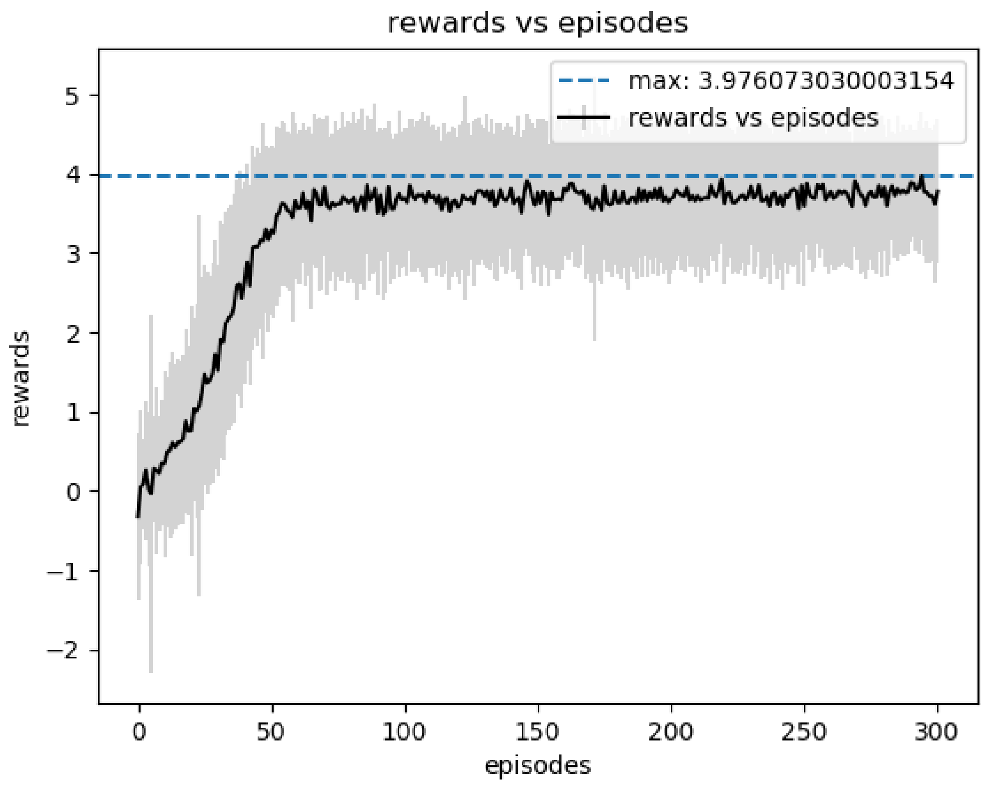

Figure 1: Softmax Sarsa Lamda Gridworld

- Softmax Q Lamda Gridworld
- $lr = 0.1$ $sigma = 5$ $lambda = 0.5$

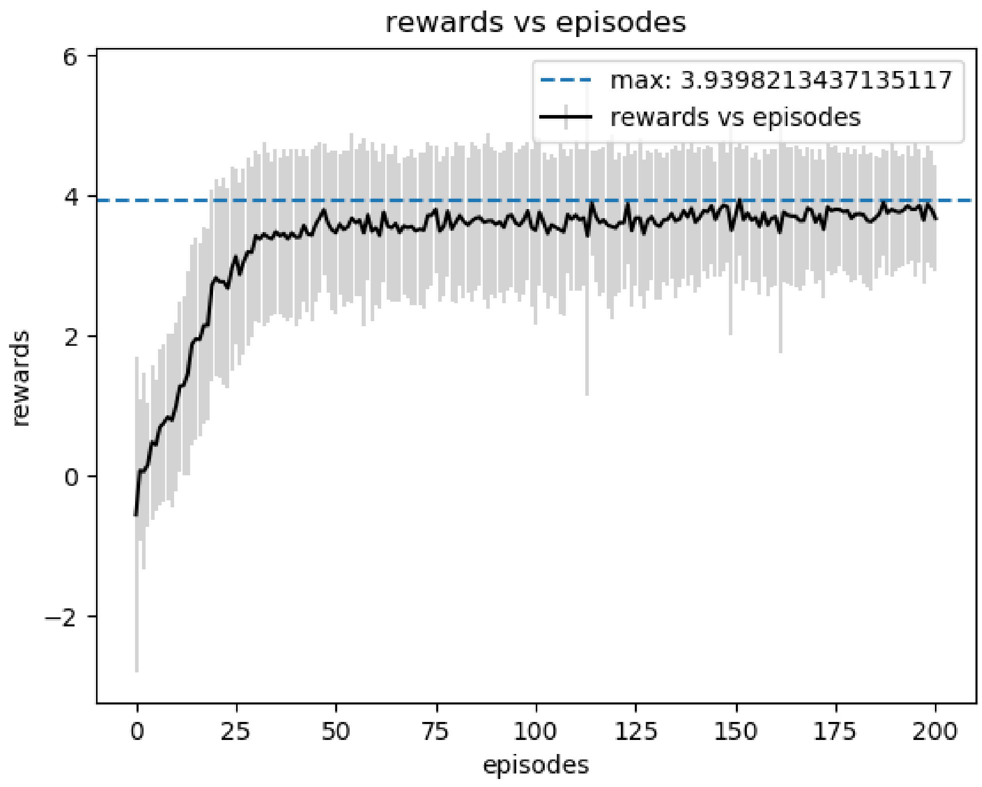

Figure 2: Softmax Q Lamda Gridworld

- Softmax Sarsa lamda Mountain Car Hyperparameters
- $lr = 0.00316$ $sigma = 0.0001$ $order = 3$ $lamda = 0.5$

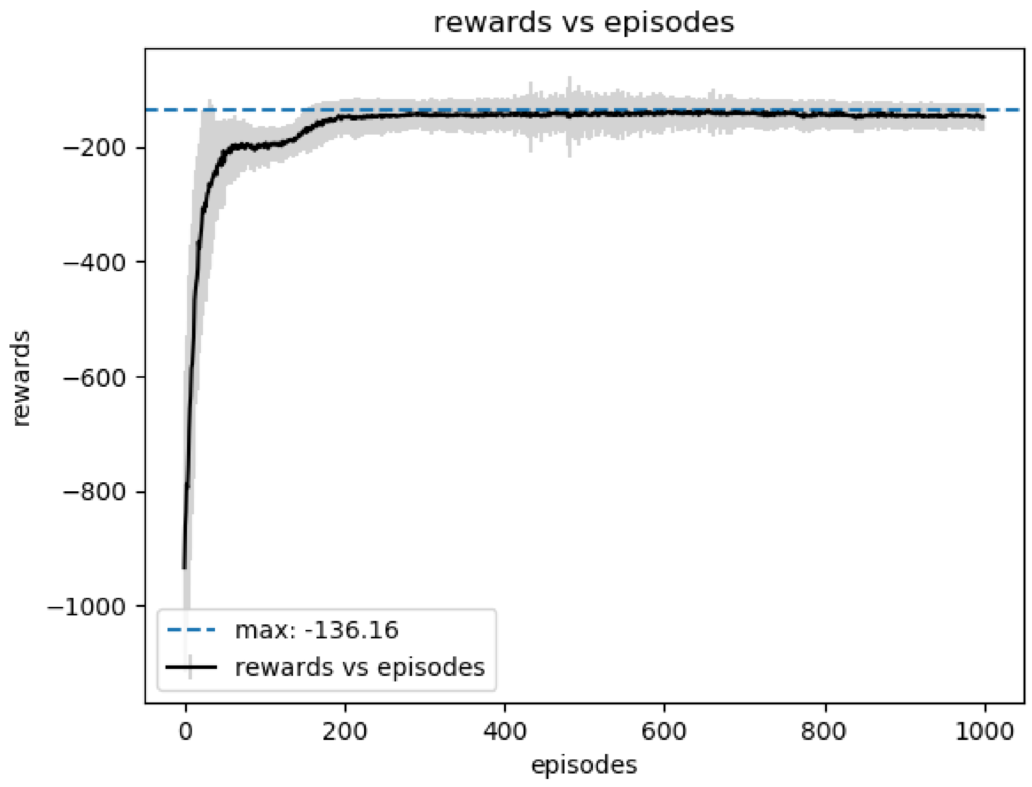

Figure 3: Softmax Sarsa lamda Mountain Car

- Softmax Q lamda Mountain Car Hyperparameters
- $lr = 0.00316$ $sigma = 0.0007$ $order = 2$ $lamda = 0.1$

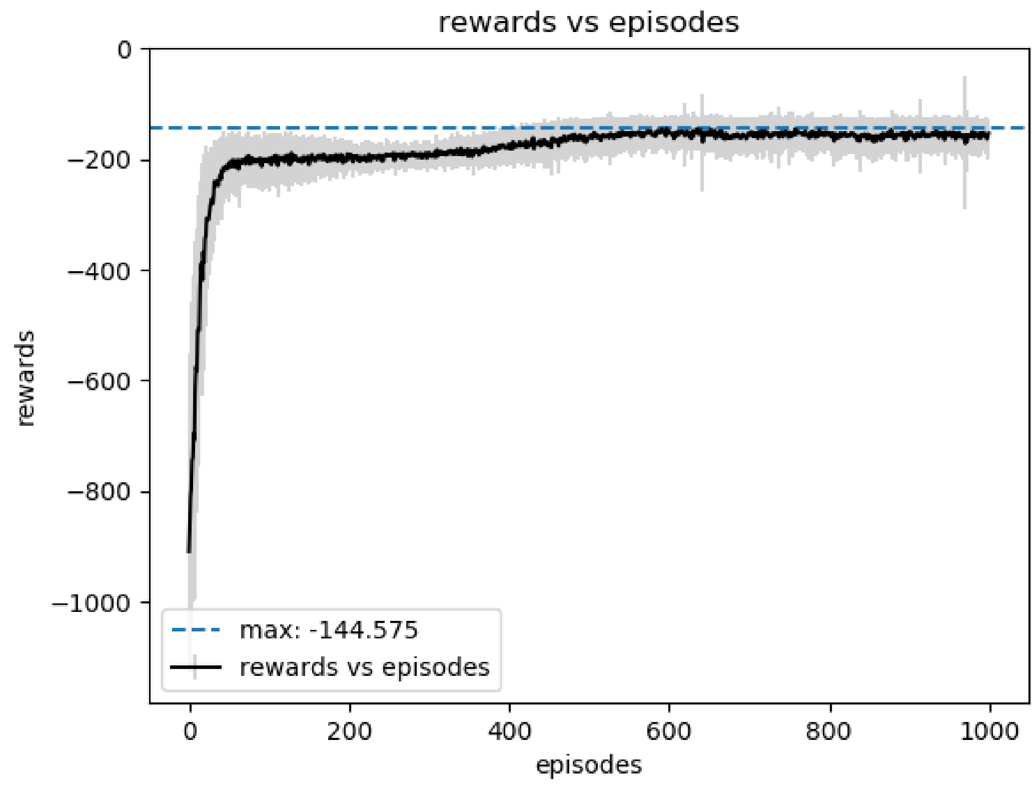

Figure 4: Softmax Q lamda Mountain Car

- Comments: Q λ and Sarsa λ seemed to perform slightly faster than the non-λ variants in gridworld probably because e_trace attributes the td_error not only to the current state but also to states visited so far in the episode and hence it helps in learning in the case of sparse rewards. The results of λ and non-λ algorithms are similar and since Sarsa and Q Learning were already giving good results, it is difficult to compare them with Sarsa λ and Q λ. The hyperparameters used in QLearning and Sarsa worked with Sarsa λ and Q λ and it was easier to get good results with slight tuning.

- Solution 4:
- Softmax Actor Critic GridWorld Hyperparameters
- Actor $lr = 0.1$ $sigma = 10$ $lambda = 0.2$
- Critic $lr = 0.1$ $sigma = 10$ $lambda = 0.2$

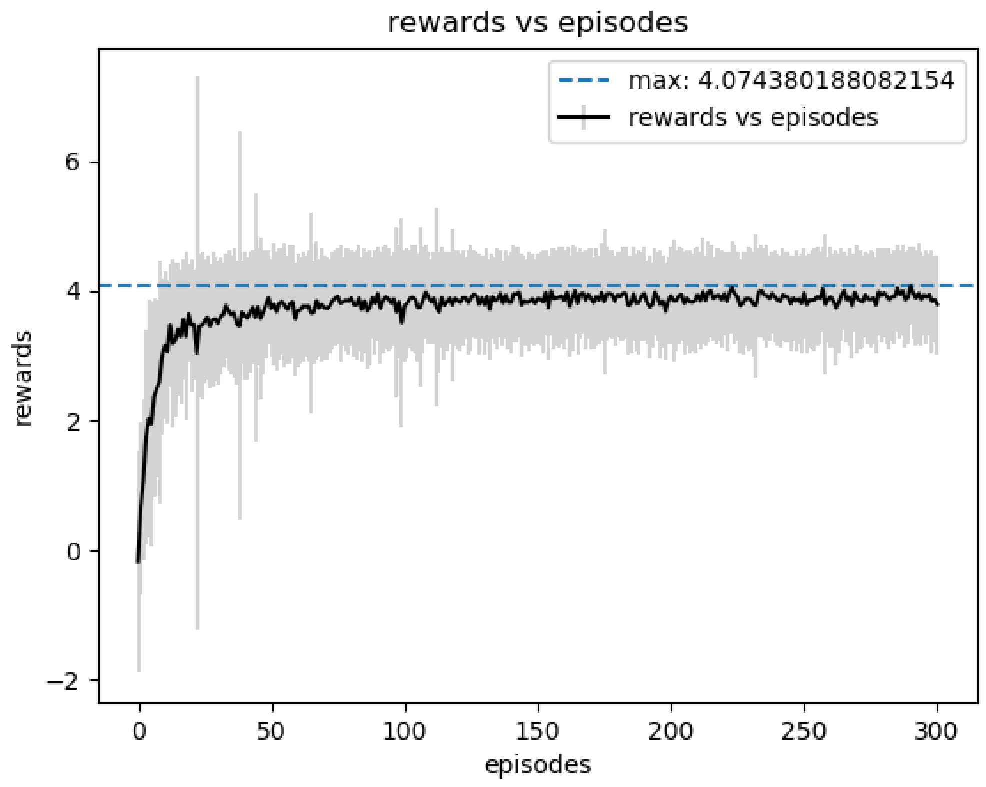

Figure 5: Softmax Actor Critic GridWorld

- Softmax Actor Critic Mountain Car Hyperparameters
- Actor $lr = 0.0005$ $sigma = 10$ $order = 2$ $lamda = 0.2$
- Critic $lr = 0.0005$ $sigma = 10$ $order = 2$ $lamda = 0.2$

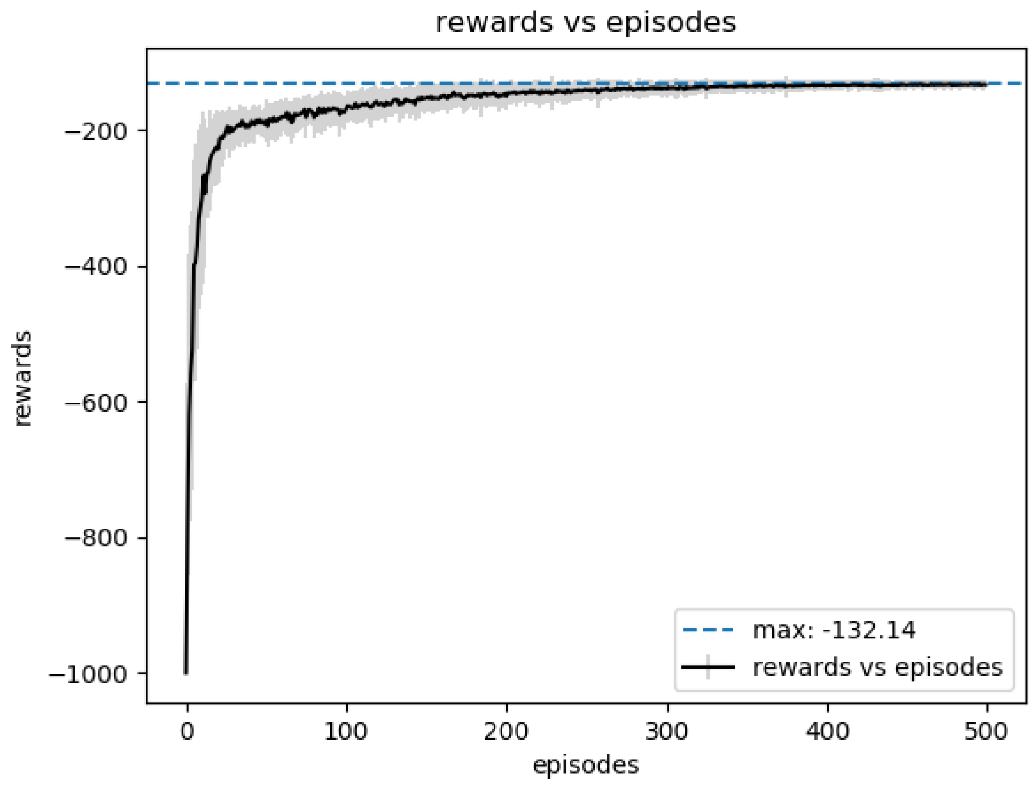

Figure 6: Softmax Actor Critic Mountain Car

- Comments : I used linear function approximator with fourier basis for both actor and critic part. Actor critic gave better results than Q-learning and Sarsa for gridworld and comparable results for cartpole. Also it seems to perform better than TD lamda and Q lamda in the case of gridworld and cartpole. Additionally, it took less time to converge in both domains. Actor critic Mountain car model converged in 500 episodes unlike q learning and sarsa which required 1000 episodes to converge. This was probably because in actor critic, we are directly optimising policy instead of doing the same via value function which might be complex to model. The problem with value-based methods is that they can have a big oscillation while training. This is because the choice of action may change dramatically for an arbitrarily small change in the estimated action values. On the other hand, with policy gradient, we just follow the gradient to find the best parameters. We see a smooth update of our policy at each step. It took a few extra hours to tune it since we had to find the right set of hyperparameters for critic as well as actor. Used tabular function approximator for gridworld actor and critic and it was relatively easier to tune.

- Solution 5:
- Softmax Reinforce GridWorld Hyperparameters
- $lr = 0.05$ $sigma = 1$ $lambda = 0.2$

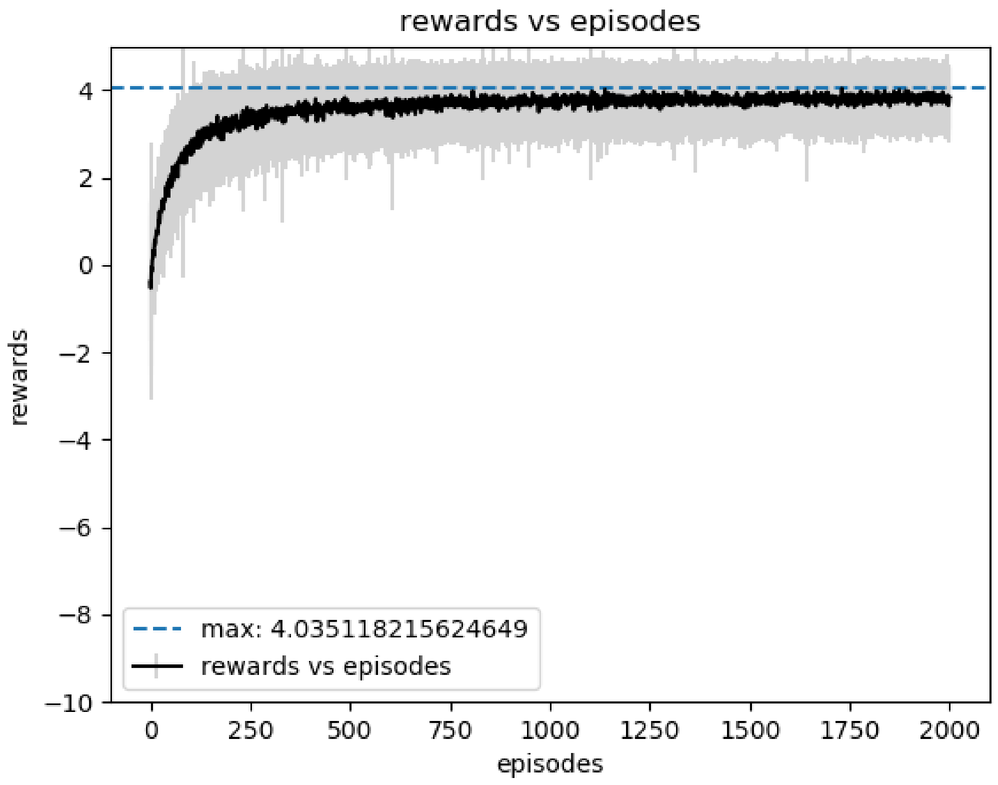

Figure 7: Softmax Reinforce without Baseline GridWorld

- Softmax Reinforce Mountain Car Hyperparameters
- $lr = 0.001778$ $sigma = 0.0001$ $order = 3$ $lamda = 0.2$ $capped\_at = 50000$

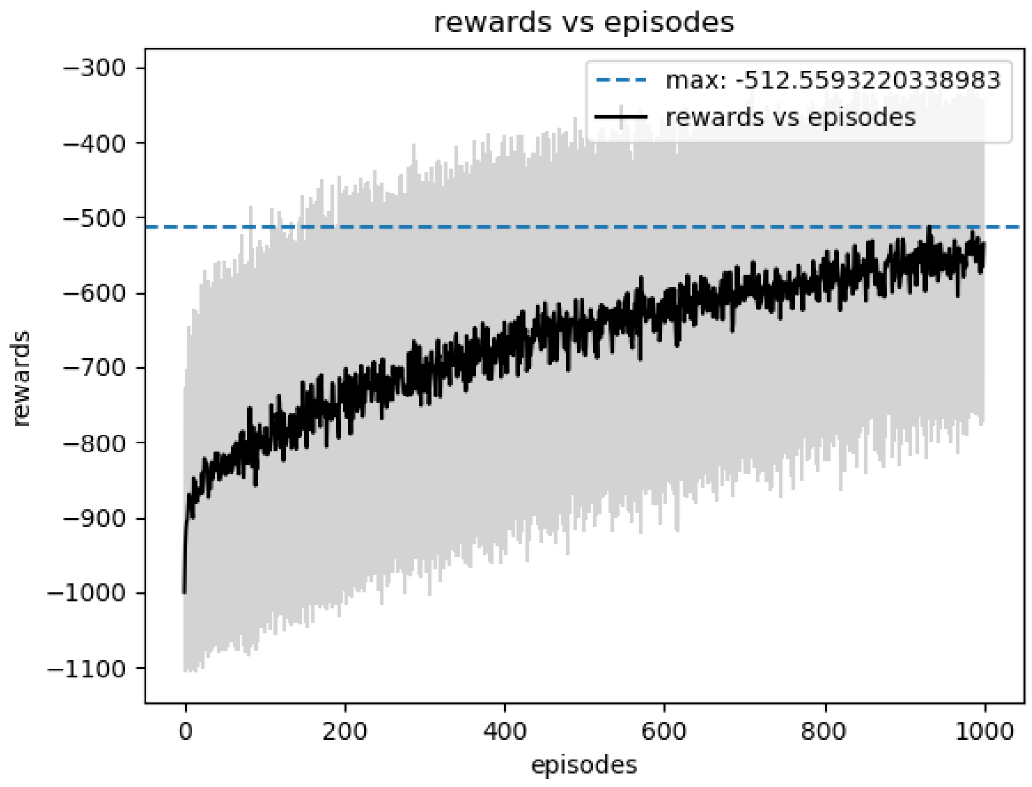

Figure 8: Softmax Reinforce Mountain Car: 59/100 trials successful

- Softmax Reinforce Mountain Car Hyperparameters
- $lr = 0.001778$ $sigma = 0.0001$ $order = 3$ $lamda = 0.2$ $capped\_at = 50000$

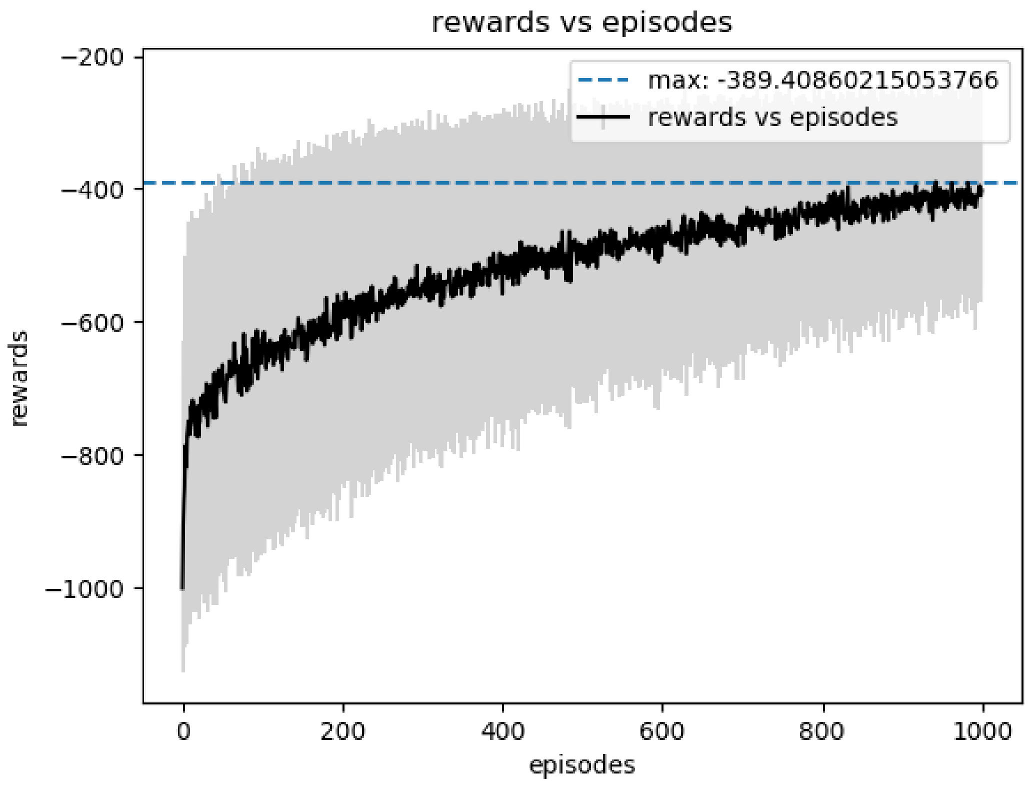

Figure 9: Softmax Reinforce Mountain Car: 93/200 trials successful

- Softmax Reinforce Mountain Car Hyperparameters
- $lr = 0.001778$ $sigma = 0.0001$ $order = 3$ $lamda = 0.2$ $capped\_at = 50000$

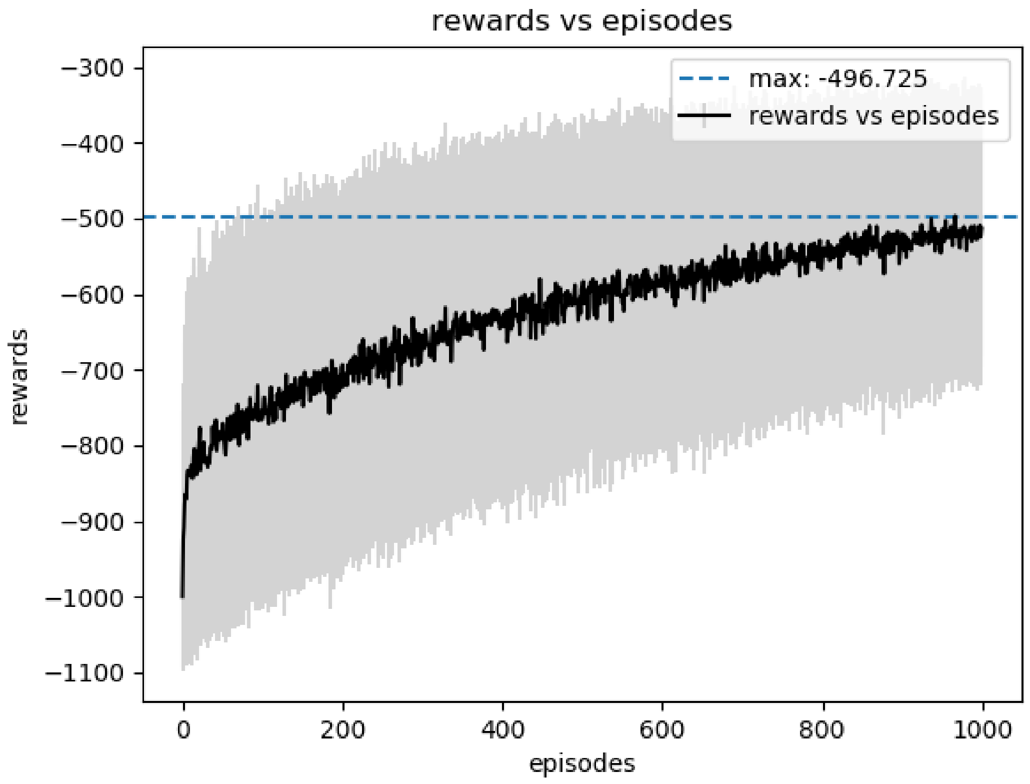

Figure 10: Softmax Reinforce Mountain Car 120/200 trials successful

- Softmax Reinforce Mountain Car Hyperparameters
- $lr = 0.000000001778$ $sigma = 0.0001$ $order = 3$ $lamda = 0.2$ $capped\_at = 50000$

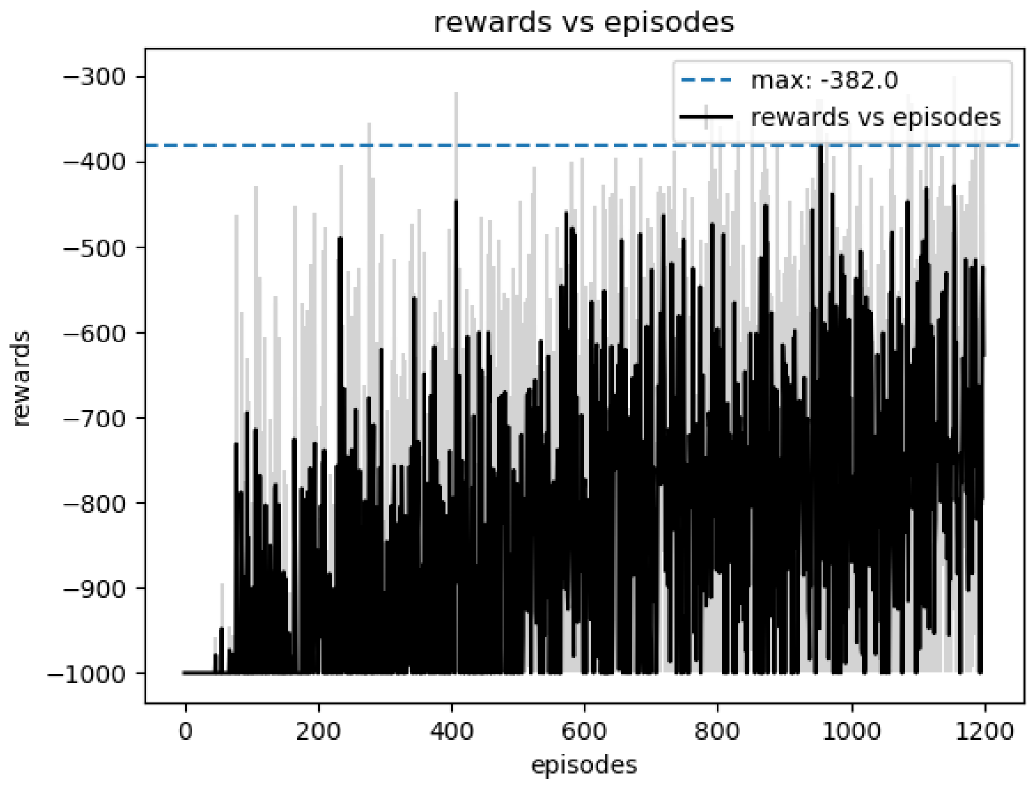

Figure 11: Softmax Reinforce Mountain Car: Consecutive 5 trials

- Please note that in first three plots (Figure 8,9,10) of REINFORCE Mountain car, trials are not consecutive and plots have not been plotted over 93/200 trials, 120/200 trials and 59/100 trials. I have discarded trials in which there is no learning at all in the first 10 episodes. This is equivalent to resetting the weights in a trial. The reason for doing so is that - if the mountain car does not see the goal state in the first few episodes (because of the limit on the no of time steps), it is very unlikely it will learn in later episodes since its weights would be adjusted in completely wrong direction. This was observed while experimenting with REINFORCE. I tried lowering the learning rate and removing the limit on the time steps to avoid trials which don't learn at all but it made the learning very slow. Hence I only plotted 5 trials which is shown in figure 11 (Rewards are clipped to 1000 to get a clear plot). Hence it made sense to discard/reset trials in which the goal state is not observed in the initial state and there is no learning (unsuccessful trials). I am capping the maximum no of timesteps the episode can run for (50000) since sometimes it gets stuck in seemingly infinite loop (large no of time steps and hence expensive updates). Discarding trials with no learning in initial phase does not affect the policy evaluation as we are NOT discarding unsuccessful episodes within trials and hence this only helps with exploration. Once the agent sees the goal state, it starts getting developing a sense of direction to proceed through trial and error and that's what I am trying to simulate here with the low computation power that I have at my disposal. Since it was taking extremely long time to train, I have only run 200 trials in total out of which a fraction (120/200) were successful. Optimistic initialization did not help. Adding large penalty at the capped time time step did not help either. I also tried normalising the gradient accumulated inorder to avoid exploding gradient problem but it did not help much in learning. So only in those episodes, in which it reaches the goal state before it reaches the time step limit , it is able to connect the dots and learn which makes sense. I think if we use shaping by demonstration strategy (which basically uses demonstration trajectory to increase potential of state,action pairs visited in the demonstration), it should be able to reach the goal faster and hence be able to learn once it consecutively sees goal state over few episodes. Did not try this out due to lack of time. Also tried reinforce with baseline for a large range of hyperparameters but could not achieve better results. Used linear function approximator with fourier basis for mountain car and it took several hours to tune reinforce since updates are made at the end of episode and some episodes take a very long time in the initial stage of learning. Used tabular function approximator for gridworld and it was relatively easier to tune.