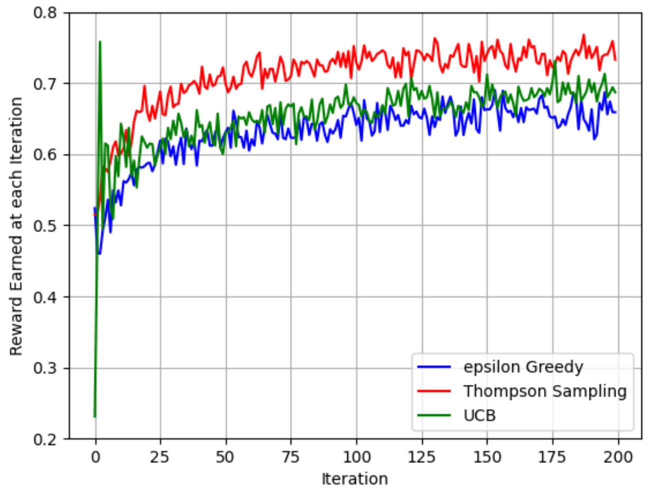
CS885 Assignment 2

Angelo Rajendram

October 2022

1 Part 1 Write Up



The figure above displays the reward earned at each iteration averaged over 1000 trials of 200 iterations. Here we see that all three methods converge to the best arm, corresponding to a 0.75 probability of earning a reward. The Thompson sampling bandit appears to converge quickest, followed by the Upper-Confidence-Bound (UCB) bandit, and then the ϵ-greedy bandit.

1.1 ϵ-greedy bandit

All of these methods force exploration because there is uncertainty in the value estimates of each action. The greedy actions look best based on the reward history, but other actions may actually be better. ϵ-greedy action selection forces non-greedy actions to be tried, however, it does so without preference for their relative optimality or uncertainty. This explains its relatively slow convergence rate.

1.2 UCB bandit

Upper-Confidence-Bound action selection considers the uncertainties in the reward estimates for each action and how close these estimates are to being optimal. UCB asks how far away the empirical mean $\tilde{R}(a)$ is from the true mean $R(a)$.

$$\text{Knowing: } |R(a) - \tilde{R}(a)| \leq \mathit{bound},$$

$$\text{implies that: } R(a) \leq \tilde{R}(a) + \mathit{bound}$$

And so it selects the arm with the largest $\tilde{R}(a) + \mathit{bound}$. As more samples are collected, the reward estimate bound for each arm becomes tighter as the uncertainty reduces.

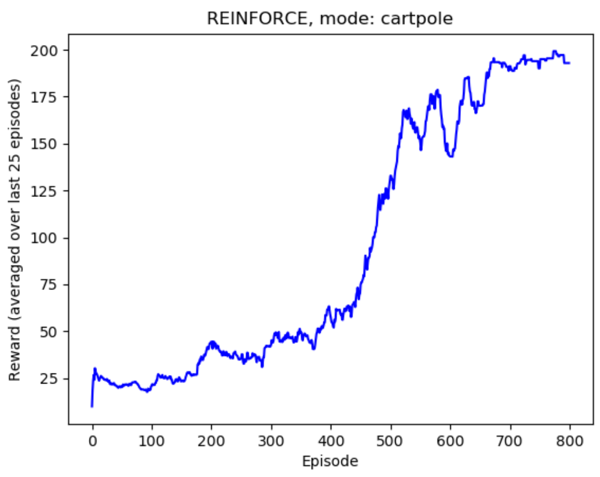
Interestingly, a spike is observed over the first few steps for UCB. At the start, none of the actions have been explored and so the upper confidence bound is infinite. In the first few iterations, the agent selects the actions randomly. When it encounters a high reward, the estimated reward for that action increases significantly since there is only a single sample. Consequently, the agents will make a greedy selection at the same time step across all the trials resulting in the spike. Even at later iterations the agent may still choose the worst arm and cause a lower score than at the beginning. When the total number of iterations is increased to a large value (such as 1000 in this case), the UCB score will converge to the value of 0.75.

1.3 Thompson sampling bandit

Thompson sampling is a Bayesian approach where the uncertainty of rewards for each action is expressed with a prior distribution which is then used to compute a posterior distribution as rewards are collected from the environment. Samples are taken from the distribution associated with each action, and the action with the highest empirical mean of these samples is selected. Since the action selection is based on samples from the prior distributions, it is inherently stochastic. As the number of samples taken from this prior distribution increases, the empirical mean converges to the expected value of the prior distribution, which results in more deterministic action selection. In other words, the agent will explore less as more samples are taken from the prior distribution. For the agent used to generate the plot above, only one sample per action was taken from the prior distribution, and so it tends to explore more relative to an agent that takes more samples from this distribution.

2 Part 2 Write Up

2.1 REINFORCE



The REINFORCE algorithm is an example of a stochastic policy gradient method. This method is a model-free policy-based method without an explicit value function representation. Some of the stochasticity in the plot is a result of using only one episode per epoch in this assignment. If multiple episodes had been collected and updates performed on a batch of episodes (applying the average gradient), smoother updates would be possible. The policy learned through the REINFORCE algorithm improves significantly around episode number 500, and when the cart has learned to balance the pole, it collects a reward at every time step in the episode accumulating a total reward of 200 at which point the episode terminates.

With a stochastic policy parameterized by θ, the gradient descent update rule at each episode of length T for the REINFORCE algorithm used in the assignment is as follows,

$$\theta \leftarrow \theta + \alpha \sum_{n=0}^{T} \gamma^n G_n \nabla_\theta \log \pi_\theta(a_n|s_n)$$

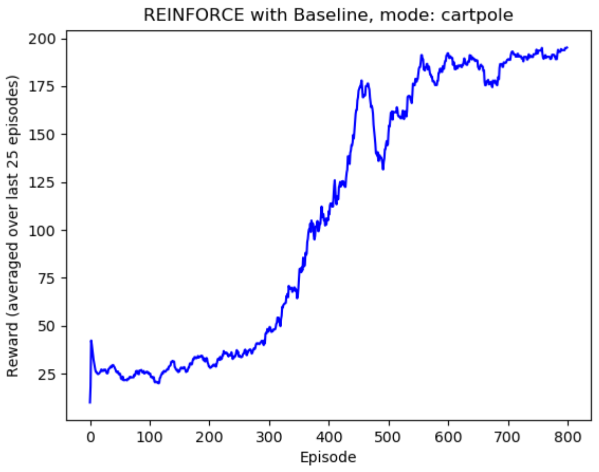
Where G_n is the accumulated return at each time step n,

$$G_n = \sum_{t=0}^{T-n} \gamma^t r_{n+t}$$

This update rule is derived from the Stochastic Gradient Policy Theorem,

$$\nabla_\theta V_\theta(s_0) \propto \sum_s \mu_\theta(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q_\theta(s,a)$$

2.2 REINFORCE with a baseline



The REINFORCE algorithm with a baseline is a model-free, policy and value-based method. Since both policy and value representations are used, this is similar to an actor-critic technique, though the state-value function is not used for bootstrapping. The equivalent Stochastic Gradient Policy Theorem with a baseline $b(s)$, is as follows,

$$\nabla_\theta V_\theta(s_0) \propto \sum_s \mu_\theta(s) \sum_a \nabla_\theta \pi_\theta(a|s)[Q_\theta(s,a) - b(s)]$$

Note that,

$$\sum_a \nabla_\theta \pi_\theta(a|s) b(s) = b(s) \nabla_\theta \sum_a \pi_\theta(a|s) = b(s) \nabla_\theta 1 = 0$$

The baseline is chosen to be some approximation of the value function $V^\pi(s)$. Let the Advantage be defined as follows,

$$A(s,a) = Q(s,a) - V^\pi(s)$$

The difference between the Value function and the Q function is informative of the gain or loss that results from choosing a particular action. If the Advantage function is chosen to replace G_n (the sum of rewards going forward from time step n until the end of the episode at T), it eliminates much of the stochasticity due to actions at subsequent time steps. This results in faster empirical convergence relative to the vanilla REINFORCE algorithm. Looking at the plot we see that there is a significant improvement in the learned policy around episode 400, several episodes earlier than in the plain REINFORCE case.

For REINFORCE with a baseline in the assignment, the update rule is as follows (here δ is the Advantage),

$$\delta \leftarrow G_n - V_w(s_n)$$

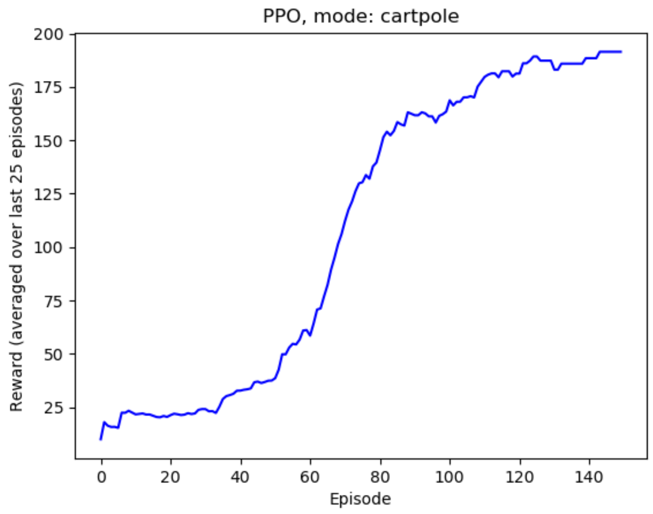
Update the Value function at each step in an episode,

$$w \leftarrow w + \alpha_w \delta \nabla_w V_w(s_n)$$

Update the policy network at the end of each episode,

$$\theta \leftarrow \theta + \alpha_\theta \sum_{n=0}^{T} \gamma^n \delta \nabla_\theta \log \pi_\theta(a_n|s_n)$$

2.3 Proximal Policy Optimization (PPO)



For the previous techniques using policy gradient methods, it isn't certain how best to set learning rate α. Setting a small α results in reliable but slow convergence, on the other hand, setting a larger α results in fast but unreliable convergence. Furthermore, the use of surrogate objectives (such as an approximation of the value, V) implies that it may be trustable only in a small region. The principle behind trust region policy optimization (TRPO) and proximal policy optimization (PPO) is to limit the search to a small trust region. The update step for TRPO is as follows,

$$\theta \leftarrow \underset{\tilde{\theta}}{\operatorname{argmax}} \; E_{s_0 \sim p}[V^{\pi_{\tilde{\theta}}}(s_0) - V^{\pi_\theta}(s_0)]$$

Subject to the constraint of the KL-Divergence, D_{KL},

$$\max_s D_{KL}(\pi_\theta(\cdot|s), \pi_{\tilde{\theta}}(\cdot|s)) \leq \delta$$

Since TRPO is conceptually and computationally challenging in large part due to the optimization constraint, PPO improves upon this by recognizing that the constraint effectively limits the ratio,

$$\frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)}$$

The optimization step for PPO is given by,

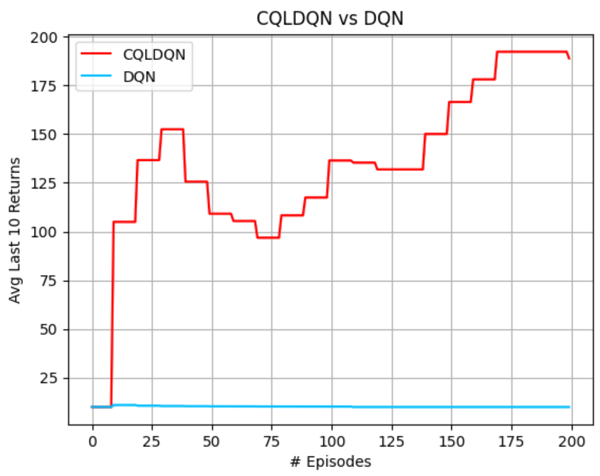
$$\theta \leftarrow \underset{\tilde{\theta}}{\operatorname{argmax}} \frac{1}{N} \sum_{n=0}^{N-1} \min\left\{ \frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)} A(s_n, a_n),\ \ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)}, 1-\epsilon, 1+\epsilon \right) A(s_n, a_n) \right\}$$

where the $\underset{\tilde{\theta}}{\operatorname{argmax}}()$ is computed by taking several gradient ascents steps in practice.

Using PPO results in a much quicker convergence of the policy compared to the previous methods since it adaptively changes its learning rate. Here, the plot is shown for 150 episodes whereas 800 episodes were shown in the previous two methods.

3 Part 3 Write Up

3.1 Conservative Q-Learning (CQL)



Offline DQN learns a policy based on data and cannot make corrections by interacting with the environment. Off-policy Q Learning consists of a policy evaluation step,

$$Q^{\pi} = \text{argmin}_Q\, E_{(s,a,r,s')\sim D}\left[\left(r + \gamma E_{a'\sim\pi(a'|s')}[Q(a',s')] - Q(s,a)\right)^2\right]$$

followed by greedy policy improvement,

$$\pi_{k+1}(s) \leftarrow \text{argmax}_a\, Q^{\pi_k}(s,a) \quad \forall s$$

At every update step, the Q values for some state-action pairs are overestimated and others are underestimated. Since the policy is updated by selecting the greedy action according to the Q-values, the actions with over-estimated Q-values are likely to be selected during training. In other words, the DQN technique is very sensitive to distribution shift. If the policy learned by an offline DQN agent encounters a state that wasn't frequently observed in the training data, its Q-value estimate for the actions at that state pair may be significantly off-base, and it can select sub-optimal actions that result in poorly performing policies when implemented in the actual environment.

Conservative Q-Learning (CQL) mitigates these issues by introducing a penalty term that explicitly tries to minimize the Q-values during policy evaluation,

$$\hat{Q}^{\pi} = \text{argmin}_Q \, \eta E_{s\sim D, a\sim\pi(a|s)}[Q(s,a)] + E_{(s,a,r,s')\sim D}\left[\left(r + \gamma E_{a'\sim\pi(a'|s')} Q(s',a') - Q(s,a)\right)^2\right]$$

where η determines the importance of the penalty.

This penalty term can be improved further for state-action pairs that were seen frequently in the training data as follows,

$$\hat{Q}^{\pi} = \text{argmin}_Q \eta (E_{s\sim D, a\sim\pi(a|s)}[Q(s,a)] - E_{s,a\sim D}[Q(s,a)]) + E_{(s,a,r,s')\sim D}\left[\left(r + \gamma E_{a'\sim\pi(a'|s')} Q(s',a') - Q(s,a)\right)^2\right]$$

This is done since there is no need to penalize state-action pairs that have been seen often in the training data, only those state-action pairs that were infrequently visited should be penalized to mitigate distribution shift. If the support(π) $\subseteq$ support(π_β), where π_β is the policy used to collect the training data, then for a sufficiently large η,

$$\tilde{V}^{\pi}(s) \leq V^{\pi}(s) \;\; \forall s \in D$$

where,

$$V^{\pi}(s) = E_{a\sim\pi(a|s)} Q^{\pi}(s,a)$$

The Value function is lower bounded by this heuristic, and in the case of the greedy policy with respect to Q values,

$$\pi(s) = \text{argmax}_a \, Q(s,a), \text{ and,}$$

$$\tilde{Q}^{*} = \text{argmin}_Q \eta \left(E_{s\sim D}\left[\max_a Q(s,a)\right] - E_{s,a\sim D}[Q(s,a)]\right) + E_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)^2\right]$$

Using this penalty term produces a Q function and thereby a policy that is more stable in practice. This can be seen in the plot where the policy learned with DQN does not generalize well, and cannot keep the pole balanced, however, the policy learned with CQL succeeds at the cart-pole task.