Reinforcement Learning Assignment 1

Student Number:13064947 - John Goodacre

February 15, 2018

1 Introduction

This file gives shows the output charts from the accompanying Jupyter notebook for the first Reinforcement Learning coursework.

2 Reinforce

Code was written to implement 'greedy', 'ϵ-greedy', and 'UCB'. This part of the question addressed 'Reinforce'.

Write down the update function to the preferences for all actions $\{a_1, \ldots, a_n\}$ if you selected a specific action $A_t = a_i$ and received a reward of R_t.

In other words, complete:

$$\begin{aligned} p_{t+1}(a) &= \ldots && \text{for } a = A_t \\ p_{t+1}(b) &= \ldots && \text{for all } b \neq A_t \end{aligned}$$

Here the notes use H for preferences, rather than p...but following the notation given above.

Note $\pi(a) = \frac{e^{p_t(a)}}{\sum_c e^{p_t(c)}}$ is our softmax function, with c ranging over all of our preferences. The question doesn't ask for workings and we are time-consumed, but just to show I know whats going on.

So our update for softmax using the log-likelihood trick is

$\mathrm{p}_{t+1}(a) = p_t(a) + \alpha \frac{\partial \log \pi_t(A_t)}{\partial p_t(a)}$

$\mathrm{p}_{t+1}(a) = p_t(a) + \alpha \frac{\partial \log \pi_t(A_t)}{\partial \pi_t(A_t)} \frac{\partial \log \pi_t(A_t)}{\partial p_t(a)}$

$\mathrm{p}_{t+1}(a) = p_t(a) + \alpha \frac{1}{\pi_t(A_t)} \frac{\partial \log \pi_t(A_t)}{\partial p_t(a)}$ **(***)**

Now if we look look at the last term of (***),

$$\frac{\partial}{\partial p_t(a)}\left(\log \pi_t(A_t)\right) = \left(\sum_c e^{p_t(c)}e^{p_t(A_t)} - e^{p_t(A_t)}e^{p_t(a)}\right)\frac{1}{\left(\sum_c e^{p_t(c)}\right)^2}, \text{ if } a = A_t$$

and,

$$\frac{\partial}{\partial p_t(a)}\left(\log \pi_t(A_t)\right) = \left(-e^{p_t(A_t)}e^{p_t(a)}\right)\frac{1}{\left(\sum_c e^{p_t(c)}\right)^2}, \text{ if } a \neq A_t$$

which simplifies to

$$\frac{\partial}{\partial p_t(a)}\left(\log \pi_t(A_t)\right) = \pi_t(A_t) - \pi_t(A_t)\pi_t(a), \text{ if } a = A_t$$

$$\frac{\partial}{\partial p_t(a)}\left(\log \pi_t(A_t)\right) = -\pi_t(A_t)\pi_t(a), \text{ if } a \neq A_t$$

Plugging these into (***) gives our final updates.

$$\begin{aligned} p_{t+1}(a) &= p_t(a) + \alpha R_t\left(1 - \pi_t(a)\right) && \text{for } a = A_t \\ p_{t+1}(b) &= p_t(b) - \alpha R_t \pi_t(b) && \text{for all } b \neq A_t \end{aligned}$$

Where α is our learning rate.

3 First experiment - positive rewards

Reinforce with and without a baseline was implemented and the algorithms tested in experiment 1 (positive rewards). The results as in fig 1.

3.1 Name the best and worst algorithms

One might think random is the worst, however ironically if the very first action taken by the greedy algo is bad, then it is perfectly feasible for this to be the worst. Greedy never explores. In the case above, greedy is the worst, although I would argue random is a pretty bad way to go also.

Of the ϵ-greedy algorithms in the short run the one with the highest exploration rate (0.1) is likely to do best (unless the latter is very lucky in locking onto a close to optimal action early), however in the long run when the algos have found optimal or close to optimal actions it will explore more than the ϵ-greedy (0.01) algorithm and thus do more damage in terms of total regret. In the long run all these have linear total regret.

In the example given above UCB has the lowest total regret. It is also logarithmic in the long run in terms of total regret, thus offering better guarantees. It is also very close in performance in the short run to ϵ-greedy (0.1). Thus, I would argue this has the strongest performance in this experiment.

The reinforce algo's however both also do very well. Similar to UCB as regards performance. The reinforce algo with baseline indeed appears to be showing the lowest current regret of all and has been 'best' for several hundred iterations. There is of course a secondary advantage in terms of policy gradient methods' ability to scale (albeit not needed in this context).

3.2 Which algorithms are guaranteed to have linear total regret?

Greedy can immediately lock onto a sub-optimal action forever and so has linear expected regret. Also due to the fact that both ϵ-greedy algorithms will always continue to explore sub-optimal actions with at least a probability of $\frac{\epsilon}{|A|}$, then they also have linear regret. A random algorithm will choose sub-optimal actions with a probability of $\frac{1-p}{|A|}$, where p is the sum of probabilities of optimal actions, thus it again has linear regret.

3.3 Which algorithms are guaranteed to have logarithmic total regret?

The theorem of 2002 by Auer at al. (presented in the lectures), proves that UCB with $c = \sqrt{2}$ has a logarithmic regret. In the example above I chose c=1.

It also appears empirically in this example that reinforce both with and without a baseline shows logarithmic regret. However, this surely is related to the particular parameterisation chosen for the policy gradient and I was unable to find any mathematical guarantees either in the lecture, the notes, Sutton and Barto or via a quick internet search. Thus I am slightly wary of claiming flat out any guarantees because I can easily imagine a poor parameterisation of the problem. Perhaps simple Bandit problems like this do have some proofs of guarantees, but my quick search didn't find them.

3.4 Which of the ϵ-greedy algorithms performs best? Which should perform best in the long run?

The ϵ-greedy algorithm with the higher exploration rate, i.e. 0.1 performs the best. However, intuitively one should realise that at early stages this is likely to outperform the ϵ-greedy algorithm with an exploration rate of 0.01 (apart from the unlikely and lucky coincidence that the latter locks onto a close to optimal choice at the beginning). It is exploring more and likely to improve its greedy choice early on.

However in the long run when it may already have discovered the optimal choice, it will unfortunately continue to explore at a fixed rate of 0.1, higher than the exploration rate of 0.01 in the latter algorithm. The algorithm with the lower rate of exploration will outperform in the long run.

4 Second experiment - negative rewards

The algorithms were rerun with negative rewards. The outputs are given in fig 2.

4.1 Explain which algorithms improved from the changed rewards, and why.

Greedy and ϵ-greedy. I think the key intuition here is about the benefit or disbenefit of exploration in this context because of the new distribution of rewards. In the previous example when all rewards were positive then we would start with a set of Q-values all equal to zero...the moment the algorithm received a positive reward then the greedy algorithm would latch onto this action for good. This logic also applied to the benefits of a higher exploration rate for the same reasons.

However, when we are minimising losses, then the moment the greedy algorithm receives a negative reward it will update a negative Q-value for this action (when the others are still zero) and so it will continue to explore. Each time this happens, it will explore and thus behaves differently due to the distribution of rewards and will perform better. However compared to greedy, at the end of the day both ϵ-greedy algos will continue to explore continually and show better long term performance than greedy (albeit with some of the benefits of the higher exploration rate now negated).

I realise I have gone above two sentences but will be brief, but as observations both reinforce algos now show very similar performance. UCB is still the best over the short-run, but there is also a difference.It explores via a mean and an uncertainty - so whereas before positive rewards would have increased means and reduced uncertainty. Here as the first negative rewards come through the distribution will be pushed to the left with less uncertainty - actually encouraging further exploration of unknown actions).

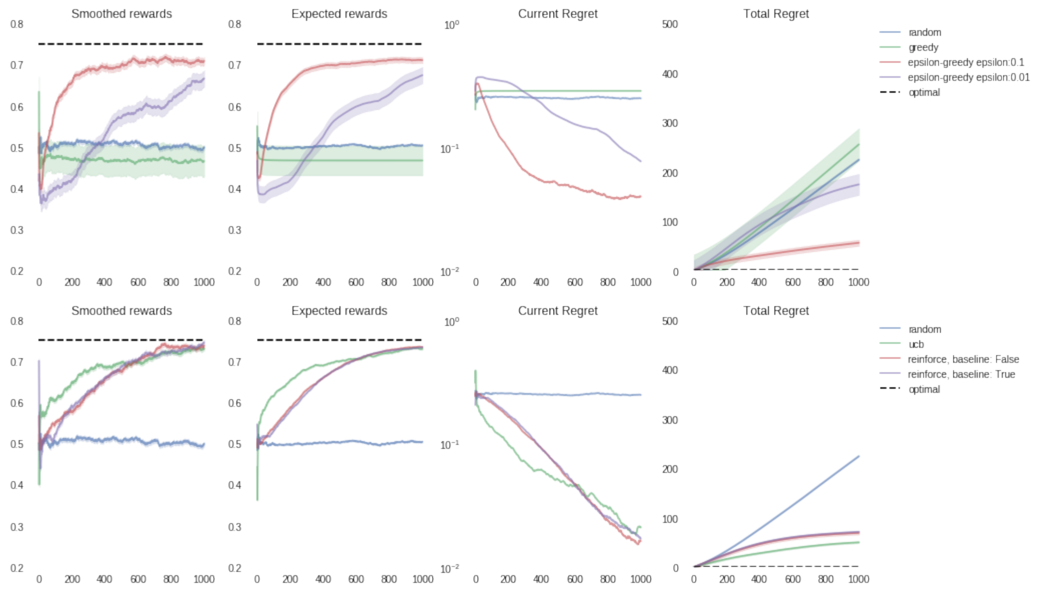

Figure 1: Bernoulli Bandit positive rewards

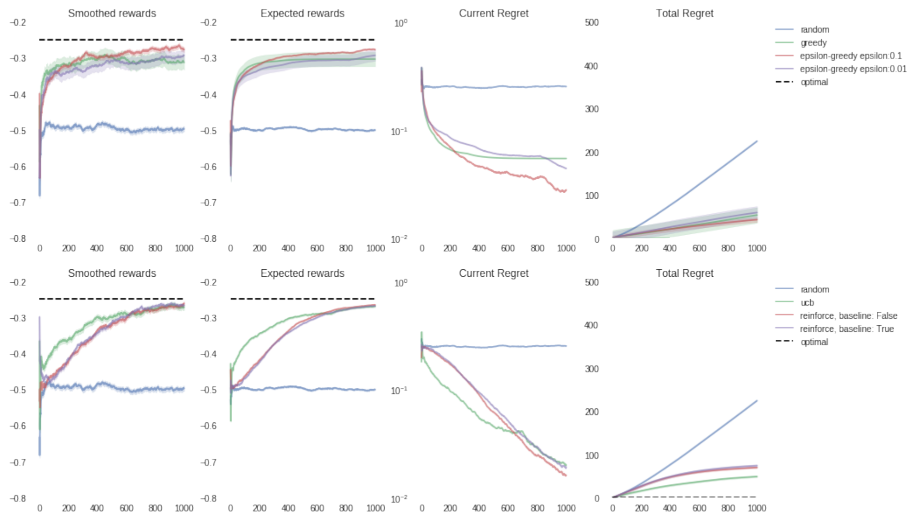

Figure 2: Bernoulli Bandit negative rewards