Reinforcement Learning With Hado van Hasselt

Yaro Kazakov

September 11, 2024

Abstract

These are study notes to prepare myself for PhD applications.

1 Lecture 2 - Exploration & Control

1.0.1 How is RL different to other types of learning?

It used training information that evaluates the actions taken rather than instructs by giving correct actions. This is what creates a need for active exploration, for an explicit search for good behavior. Purely evaluative feedback indicates how good the action taken was, but not whether it was the best or the worst action. Purely instructive feedback, indicates the correct action to take, independently of the action actually taken.

In this chapter we stude the evaluative aspect of RL in a simplified setting, one that does not involve learning to act in more than one situation.

1.1 Exploration vs Exploitation

For this lecture we consider a simple setting where the environment has only one state.

Learning agents always need to trade off two things:

1. Exploitation: Maximize performance based on current knowledge
2. Exploration: Increase knowledge. Explore new data

This tradeoff is important to achieve maximum cumulative reward (Return) over a period of time. The best long-term strategy may involve short-term sacrifices.

1.2 Formalizing the Problem - Multi-armed Bandit

1.2.1 Bandit problems are simple problems where there is only ONE state. They are part of the Tabular Solution Methods

1.2.2 Tabular Solution Methods - A large set of methods where the action space and state space are small and the exact solutions, i.e. optimal value functions can be found. In the real world most problems use Approximate Methods.

The name comes from the slot machines analogy. You pull a lever and either get a reward or don't. You might have multiple levers, hence the name Multi-armed.

1. A multi-armed bandit is a set of distributions $R_a|a\epsilon A$
2. A is a known set of actions (or "arms")
3. R_a is a distribution on rewards, given action a
4. At each time step t the agent selects an action A_t
5. The environment generates a reward $R_t \;\; R_{A_t}$

6. The goal is to maximize cumulative reward $\Sigma t_{i=1} R_i$

We want a **learning** algorithm that trades off exploration and exploitation. This algorithm won't always be optimal. We do this by learning **policy**.

1.2.3 Action-value Methods

The idea - estimate action values and use the estimates to make action selection decisions, which collectively are called action-value methods. The **action value** for action is the expected reward: $q(a) = E[R_t|A_t = a]$.
The optimal value is the maximization of q(a), hence involves optimal actions.

Regret of an action a is:

$\Delta_a = v_* - q(a)$

The regret for optimal value is 0.

We want to minimize our total regret of the whole time t. Maximizing cumulative reward = minimize total regret.

1.2.4 Algorithms discussed in this lecture

1. Greedy - use action value estimates
2. Epsilon Greedy - use action values esti ates
3. UCB - use action value estimates
4. Thompson Sampling
5. Policy Gradients

1.2.5 Action Values

$q(a) = E[R_t|A_t = a]$

A simple estimate is the average of the sampled rewards:

$Q_t(a) = sum(I(A_n = a)R_n)/sum(I(A_n = a)$ Indicator is 1 if True, 0 if False. Note that we want to estimate all the action leading UP TO time t, i.e. On fifth step, we want to have an estimate based on the previous four.

This can also be updated incrementally for better memory performance:

$Q_t(A_t) = Q_{t-1}(A_t) + \alpha * (R_t - Q_{t-1}(A_t))$ where alpha can be $\alpha = 1/N(A_t)$

This is a standard form for representing some iterative parameter. We'll see it many times in the book. The term in the parentheses is called **error** and alpha is **step size**. Note that a step size of $1/N_t$ leads to **SAMPLE-AVERAGE METHOD** which guarantees conversion of Q_a to its true value q_a. For other step sizes, such as in non-stationary problems where more weight is given to the latest rewards the conversion is NOT guaranteed.

From stochastic approximation theory we know:
$\Sigma\alpha_n(a) = \infty$ and $\Sigma\alpha_n^2(a) < \infty$

The first condition guarantees that the steps are large enough to overcome any initial condition or random fluctuations.

The second guarantees that eventually the steps become small enough to assure convergence.

1.3 The Greedy Algorithm

One of the simplest policies is greedy:
Select action with highest value: $A_t = argmaxQ_t(a)$

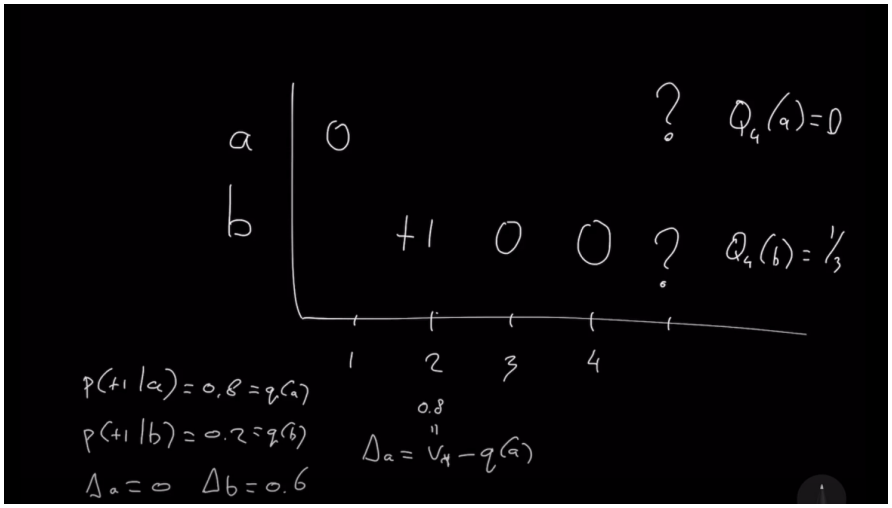

Figure 1: Regret Estimation

Greedy algorithms can get stuck on a suboptimal action forever. The total expected regret is linear. The reason it happens is because the greedy algo doesn't explore enough and gets stuck with exploitation. The alternative algo is epsilon-greedy.

1.4 ε-Greedy Algorithm

Select a greedy action with 1 - ε probability. $a = argmaxQ_t(a)$
With probability of ϵ select a random action (including the greedy one)

$\pi_t(a) = (1-\epsilon) + \frac{\epsilon}{|A|}$ if $Q_t(a) = max_b Q_t(b)$
$\pi_t(a) = \frac{epsilon}{|A|}$ otherwise

Epsilon-greedy with constant ϵ has a linear regret

1.5 Policy Gradients

$\theta_{t+1} = \theta_t + \alpha \nabla_t E[R_t|\pi_\theta t]$

The idea is to update policy parameters such that the expected value increases. So, how can we compute this gradient?

This is called Log-likelihood trick (also known as REINFORCE):

$\nabla_t E[R_t|\pi_{\theta_t}] = \nabla_\theta \Sigma \pi_\theta(a) E[R_t|A_t = a]$

The expectation is just the action-value q(a)

$\nabla_t E[R_t|\pi_{\theta_t}] = R[R_t \frac{\nabla_\theta \pi_\theta(A_t)}{\pi_\theta(A_t)}]$

$\nabla_t E[R_t|\pi_{\theta_t}] = E[R_t \nabla_\theta \log \pi_\theta(A_t)]$

The detailed derivation can be found in the book or the video by Hado.

1.6 Gradient Bandits

Using the logic above the Gradient Bandits can be derived and thus the following is obtained:

$H_{t+1}(a) = H_t(a) + \alpha * (R_t)(1 - \pi_t(A_t))$

$H_{t+1}(a) = H_t(a) - \alpha * R_t \pi_t(A_t))$ if a not A_t

Note that H is just a scalar preference with no direct meaning. What matters is the relative value of the preferences compared to other actions. If all actions receive 1000 in preference, the algorithm does not change.

This algorithm does not guarantee convergence to an optimal policy.

A baseline can be introduced that can be subtracted from the reward. In a classical Gradient Bandit it is usually the average reward up to time t. Baselines do not change the expected updated, but the do change variance.

1.7 Upper Confidence Bounds

Estimate an upper confidence $U_t(a)$ for each action value, such that q(a) ¡= $Q_t(a) + U_t(a)$ Select action maximizing upper confidence bound (UCB)

$a_t = argmax Q_t(a) + U_t(a)$

The uncertainty should depend on the number of times $N_t(a)$ action a has been selected.

Small $N_t(a)$ means large U_t estimated value is uncertain. The vice versa is true.

Then a is only selected if either...
Q_t is large (good action)
U_t is large (high uncertainty)

The full derivation can be found in the book:
$U_t(a) = \sqrt{\frac{\log t}{2N_t(a)}}$

$$\pi_t(a) = \begin{cases} (1-\epsilon) + \epsilon/|\mathcal{A}| & \text{if } Q_t(a) = \max_b Q_t(b) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$

Figure 2: Epsilon Greedy mathematical representation

1.8 Gradient Bandits

References