THE UNIVERSITY of TENNESSEE KNOXVILLE

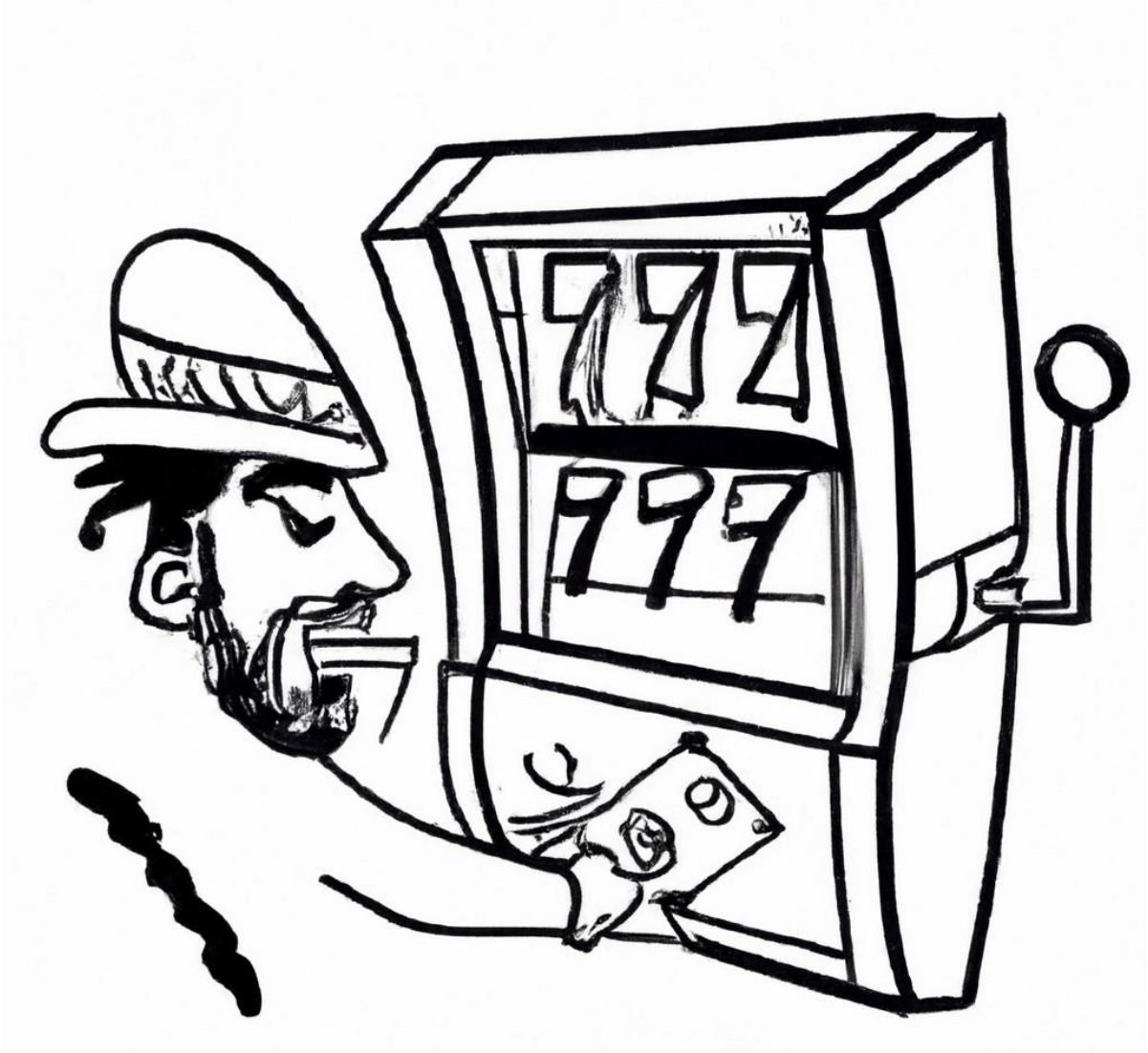
Lecture 21: Decisions and Bandits

Instructor: Sergei V. Kalinin

Mid-term

- **Multiple interesting solutions**
- **Will go in detail in the next few lectures**
- **Iterate!**

Bandit problem



- Imagine that we have a number of slot machines with different probabilities of win…
- Or different web-sites to places ads on…
- Or groups of patients for specific medical protocol….
- Or team members to synthesize certain material…
- Or reaction pathways to choose

How do we optimize this problem and maximize our reward?

Definitions:

- **Objective:** overall goal that we aim to achieve. Not available during or immediately after experiment.

- **Reward:** the measure of success available at the end of experiment

- **Value:** expected reward. Difference between reward and value is a feedback signal for multiple types of active learning

- **Action:** how can ML agent interact with the system

- **State:** information about the system available to ML agent

- **Policy:** rulebook that defines actions given the observed state

Bandits

- **Objective:** get rich!
- **Reward:** pay-off from specific hand/click-rate of ad/effectiveness of drug
- **Value:** expected reward
- **Action:** playing a hand/placing ad/administering drug
- **State:** no state
- **Policy:** gameplan given the values of specific actions

A/B Testing

- The most common exploration strategy is **A/B testing**, a method to determine which one of the two alternatives (of online products, pages, ads etc.) performs better.
- The users are randomly split into two groups to try different alternatives. At the end of the testing period, the results are compared to choose the best alternative, which is then used in production for the rest of the problem horizon.
- This approach can be applied for more then two alternatives - **A/B/n testing**.

From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

Limitations of A/B testing

- **A/B/n testing is inefficient as it does not modify the experiment dynamically by learning from the observations.** It fails to benefit from the early observations in the test by writing off/promoting an alternative even if it is obviously under- or outperforming the others.

- **It is unable to correct a decision once it's made.** There is no way to correct the decision for the rest of the deployment horizon.

- **It is unable to adapt to the changes in a dynamic environment.** If the underlying reward distributions change over time, plain A/B/n testing has no way of detecting such changes after the selection is fixed.

- **The length of the test period is a hyperparameter to tune, affecting the efficiency of the test.** If this period is chosen to be shorter than needed, an incorrect alternative could be declared the best because of the noise in the observations. If the test period is chosen to be too long, too much money gets wasted in exploration.

- **A/B/n testing is simple.** Despite all these shortcomings, it is intuitive and easy to implement, therefore widely used in practice

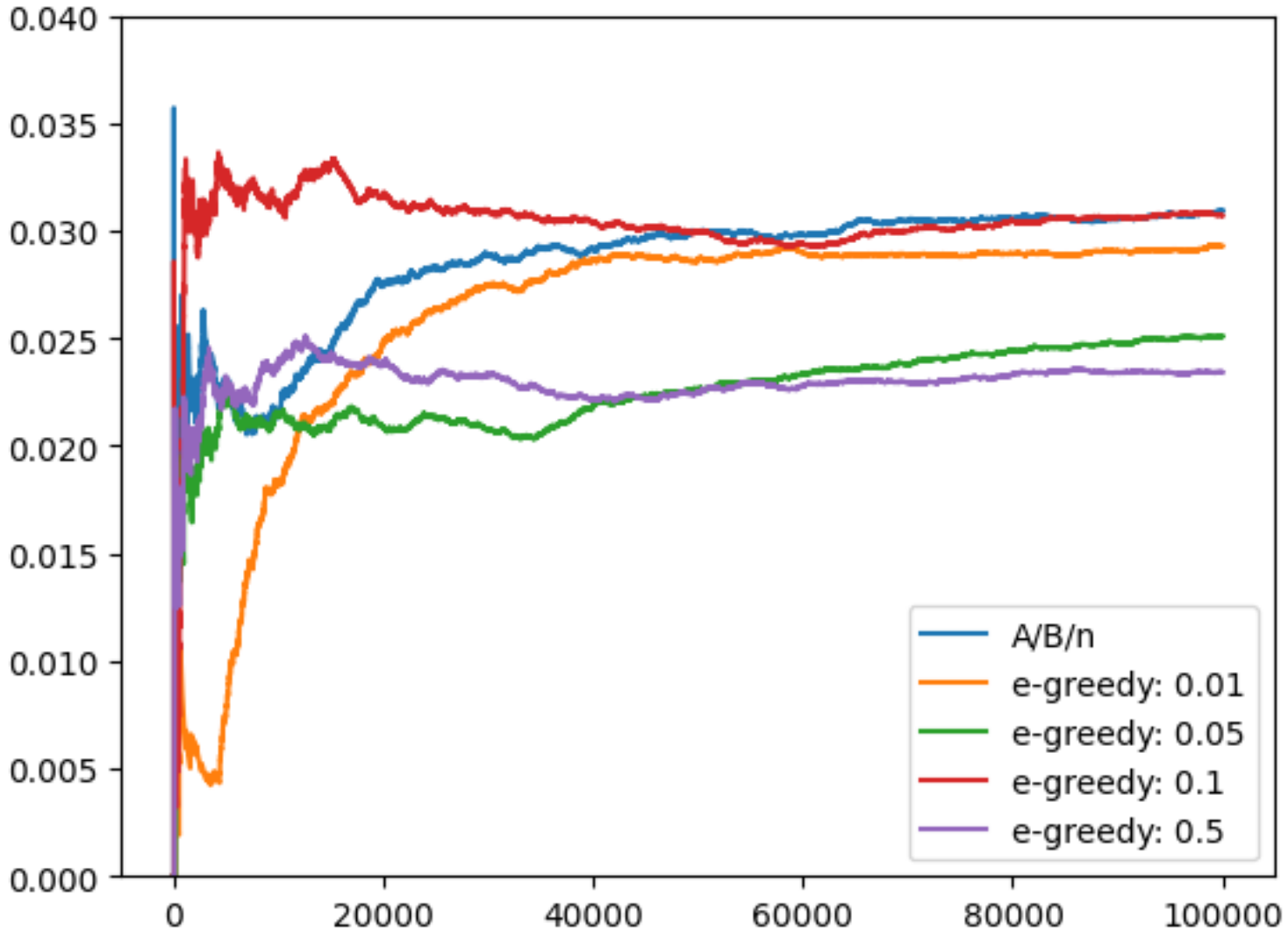
From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

ε-greedy policies

- Most of the time, greedily taking the action that is the best according to the rewards observed that far in the experiment (i.e. with 1-ε probability)
- Once in a while (i.e. with ε probability) take a random action regardless of the action performances.
- Here ε is a number between 0 and 1, usually closer to zero (e.g. 0.1) to "exploit" in most decisions.
- Obviously, the number of alternatives has to be fairly small
- Parameter ε can change during the experiment

From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

ε-greedy policies



A/B vs. ε-greedy policies

- ε-greedy actions and A/B/n tests are similarly inefficient and static in allocating the exploration budget. The ε-greedy approach, too, fails to write off actions that are clearly bad and continues to allocate the same exploration budget to each alternative. Similarly, if a particular action is under-explored/over-explored at any point, the exploration budget is not adjusted accordingly.

- With ε-greedy actions, exploration is continuous, unlike in A/B/n testing. This means if the environment is not stationary, the ε-greedy approach has the potential to pick up the changes and modify its selection of the best alternative.

- The ε-greedy actions approach could be made more efficient by dynamically changing the ε. For example, one could start with a high ε to explore more at the beginning and gradually decrease it to exploit more later. This way, there is still continuous exploration, but not as much as at the beginning when there was no knowledge of the environment.

From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

A/B vs. ε-greedy policies

The ε-greedy actions approach could be made more dynamic by increasing the importance of the more recent observations:

$$Q_{n+1}(a) = Q_n(a) + \alpha(R_n(a) - Q_n(a))$$

- **Modifying the ε-greedy actions approach introduces new hyperparameters, which need to be tuned.**
- Both gradually diminishing ε and using exponential smoothing for *Q*, come with additional hyperparameters, and it may not be obvious what values to set these to.
- Incorrect selection of these hyperparameters may lead to worse results than what the standard version would yield.

From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

Upper Confidence Bound (UCB)

$$A_t \triangleq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- At each step, we select the action that has the highest potential for reward.
- The potential of the action is calculated as the sum of the action value estimate and a measure of the uncertainty of this estimate. This sum is what we call the upper confidence bound.
- **Overall:** the action is selected either because our estimate for the action value is high, or the action has not been explored enough (i.e. as many times as the other ones) and there is high uncertainty about its value, or both.

From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

Digging deeper: UCB

$$A_t \triangleq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- $Q_t(a)$ and $N_t(a)$ have the same meanings as before. This formula looks at the variable values, which may have been updated a while ago, at the time of decision making a, whereas the earlier formula described how to update them.
- In this equation, the square root term is a measure of the uncertainty for the estimate of the action value of a.
- The more we select a, the less we are uncertain about its value, and so is the $N_t(a)$ term in the denominator.
- As the time passes, however, the uncertainty grows due to the $\ln t$ term (which makes sense especially if the environment is not stationary), and more exploration is encouraged.
- The emphasis on uncertainty during decision making is controlled by a hyperparameter, c. This obviously requires tuning and a bad selection could diminish the value in the method.

From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

Advantages and disadvantages of UCB

• **UCB is a set-and-forget approach.** It systematically and dynamically allocates the budget to alternatives that need exploration. If there are changes in the environment, for example, if the reward structure changes because one of the ads gets more popular for some reason, the method will adapt its selection of the actions accordingly.

• **UCB can be further optimized for dynamic environments, potentially at the expense of introducing additional hyperparameters.** The formula we provided for UCB is a common one, but it can be improved, for example, by using exponential smoothing. There are also more effective estimations of the uncertainty component in literature. These modifications, though, could potentially make the method more complicated.

• **UCB could be hard to tune.** It is somewhat easier make the call and say "I want to explore 10% of the time, and exploit for the rest" for the ε-greedy approach than saying "I want my uncertainty to be 0.729" for the UCB approach, especially if you are trying these methods on a brand-new problem. When not tuned, an UCB implementation could give unexpectedly bad results.

From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020