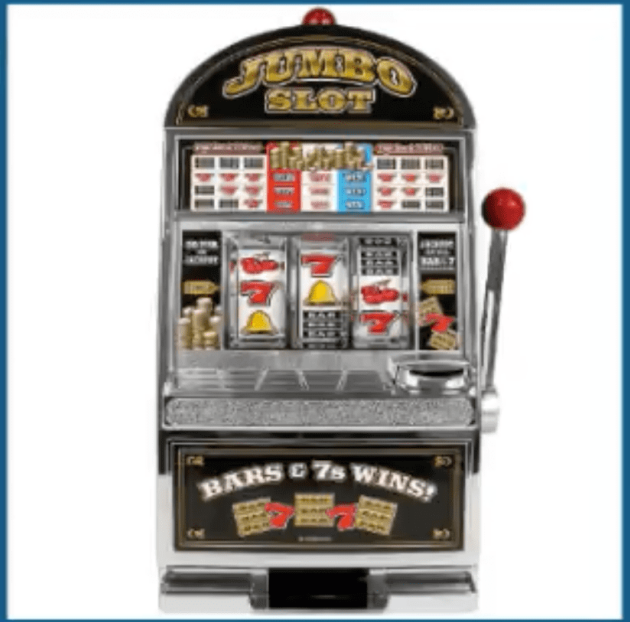
Artificial Intelligence: Reinforcement Learning in Python

By: The LazyProgrammer

Return of the Multi-Armed Bandit

- First saw in: Bayesian Machine Learning: A/B Testing
- Ex. Go to a casino; choice between 3 slot machines
- Each (for now) gives reward of 0 or 1
- Win rate is unknown
- Ex. Could be 0.1, 0.3, 0.5
- Goal: maximize your winnings
- Problem: can only discover best bandit by collecting data
- If you're psychic, you could just play the bandit with highest win rate
- Balance explore (collecting data) + exploit (playing "best-so-far" machine)



Traditional A/B Testing

- Predetermine # of times you need to play/collect data in order to establish statistical significance
- # of times needed is dependent on numerous things, like the difference between win rates of each bandit
- (But if you knew that, you wouldn't be doing this test! Weird traditional statistics ideas…)
- Important rule: Don't stop your test early
- Even if you set N=1000, but after 500 trials, you discover 90% win-rate vs. 10% win rate
- You still may not stop!

Traditional A/B Testing

- Much different from human behavior
- Suppose you're playing in a real casino, you get 2/3 on one bandit, 0/3 on the other
- You have a strong urge to play the first bandit
- Despite that their true win rates are probably not 67% and 0%
- You know "as a data scientist" that 3 is a small sample size
- As a gambler you do not "feel" that way
- In the coming lectures: we look at ways to systematically make explore-exploit tradeoff
- Better than these 2 suboptimal solutions: A/B testing, human emotion

Epsilon-Greedy

- Solution to the explore-exploit dilemma we'll use throughout the course
- Very simple
- Choose small number ε as probability of exploration
- Typical values: 5%, 10%

Epsilon-Greedy

```
p = random()
if p < eps:
  pull random arm
else:
  pull current-best arm
```

- Eventually, we'll discover which arm is the true best, since this allows us to update every arm's estimate

Epsilon-Greedy

- In the long run, allows us to explore each arm an infinite number of times
- Problem: we get to a point where we explore when we don't need to
- For epsilon=10%, we'll continue to spend 10% of the time doing suboptimal things
- Could do A/B test at some predetermined time, then set epsilon=0
- But we'll learn better ways to adapt

Estimating Bandit Rewards

- Assuming bandit rewards are not just coin tosses, what's the best way to keep track of reward?
- General method is the mean
- Works for 1/0 too
- P(k=1) = (#of1's)/(#of1's + #of0's) = (#of1's)/N
- What's the problem with this:

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

Estimating Bandit Rewards

- Need to keep track of ALL X_i
- Can we be more efficient? Yes!

Estimating Bandit Rewards

- New mean can be calculated from old mean

$$\bar{X}_N = \frac{1}{N}\sum_{i=1}^{N} X_i = \frac{1}{N}\sum_{i=1}^{N-1} X_i + \frac{1}{N}X_N = \frac{N-1}{N}\bar{X}_{N-1} + \frac{1}{N}X_N$$

$$\bar{X}_N = \left(1 - \frac{1}{N}\right)\bar{X}_{N-1} + \frac{1}{N}X_N$$

- Remember this “form”, will be very important throughout the course!

Optimistic Initial Values

- Another simple way of solving the explore-exploit dilemma
- Suppose we know the true mean of each bandit is << 10
- Pick a high ceiling as the initial mean estimate
- Before: Now:

$$\bar{X}_0 = 0 \qquad \bar{X}_0 = 10$$

- Initial sample mean is “too good to be true”
- All collected data will cause it to go down

Optimistic Initial Values

- If the true mean is 1, the sample mean will still approach 1 as I collect more samples
- All means will eventually settle into their true values (exploitation)
- Helps exploration as well
- If you haven't explored a bandit, its mean will remain high, causing you to explore it more
- In the main loop: greedy strategy only
- But using optimistic means

Optimistic Initial Values

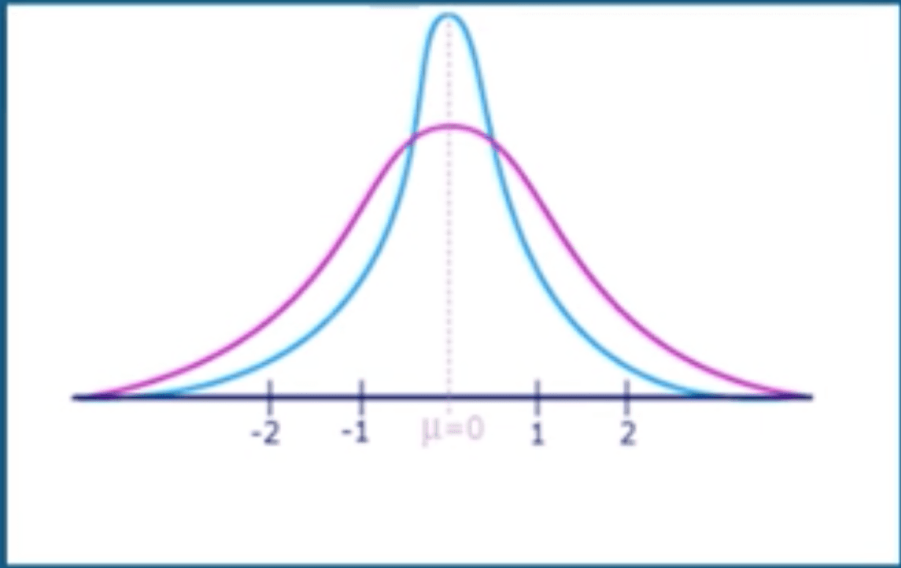
- Only 2 main code changes
- optimistic_initial_values.py

```
class Bandit:
  def __init__(self, m, upper_limit):
    self.m = m
    self.mean = upper_limit
    self.N = 1
```

```
for i in xrange(N):
  # optimistic initial values
  j = np.argmax([b.mean for b in bandits])
  x = bandits[j].pull()
  bandits[j].update(x)
```

UCB1

- Yet another method to solve explore-exploit (also from A/B testing class)
- Idea: confidence bounds
- Intuitively, we know that a sample mean from 10 samples is less accurate than a sample mean from 1000 samples



UCB1

- Chernoff-Hoeffding bound:

$$P\left\{|\bar{X} - \mu| \geq \varepsilon\right\} \leq 2exp\left\{-2\varepsilon^2 N\right\}$$

- Looks complicated, but leads to a simple algorithm!

$$X_{UCB-j} = \bar{X}_j + \sqrt{2\frac{lnN}{N_j}}$$

"Choose" epsilon equal to this upper bound

- N = number of times I've played in total, N_j = number of times I've played bandit j

UCB1

$$X_{UCB-j} = \bar{X}_j + \sqrt{2\frac{lnN}{N_j}}$$

- How do we use this formula? Same as optimistic initial values method
- Just be greedy, but with respect to $X_{UCB\text{-}j}$
- Key: ratio between ln(N) and N_j
- If only N_j is small, then upper bound is high
- If N_j is large, upper bound is small
- ln(N) grows more slowly than N_j, so eventually all upper bounds will shrink
- By then we've collected lots of data, so it's ok
- So we converge to purely greedy strategy

Pseudocode

- Simply change what goes in the argmax:

```
for n=1..N:
  j = argmax[j']{ bandit[j'].mean + sqrt(2*ln(n) / n_j') }
  pull bandit j
  update bandit j
```

- What if $N_j = 0$? Add a small number to denominator
- ucb1.py

Bayesian method

- Thompson sampling
- Let's think about confidence intervals again
- We know intuitively that sample mean from 10 samples is less accurate than sample mean from 1000 samples
- Central Limit Theorem says that sample mean is approximately Gaussian
- Any function of RVs is a RV

$$\bar{X} \sim Normal(\mu, \sigma^2/N)$$
$$X \sim Normal(\mu, \sigma^2)$$

Bayesian method

- Bayesian paradigm takes this a step further
- What if μ is ALSO a RV?
- Data is fixed; parameters are random
- I want to find the distribution of the parameter given the data
- Should be more accurate than if I did not have the data
- We call this the posterior

$$p(\theta|X)$$

Bayesian method

- Flip the posterior to find it in terms of likelihood and prior
- Disadvantage of Bayesian method: we must choose the prior
  - Non-negligible effect on posterior

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta} \propto p(X|\theta)p(\theta) = likelihood \times prior$$

- Problem: integral is usually intractable, we need “tricks” to solve this
- Special (likelihood, prior) pairs where we can easily solve this
- Called conjugate priors