Project 7 | The Arithmetic Mean and Q-Learning (Supplemental Material) #840

Steven Bryant INSTRUCTOR
18 minutes ago in Project 7 | Q-Learning Robot > – P7 | General | FAQ

PRIVATE PIN STAR WATCHING 6 VIEWS

Hey Everyone, this post is simply a view into Q-learning and how the math (or equation) for the Q-Learning update rule can be derived. Actually, this post will begin by discussing an equation that, in my opinion, is one of the most important equations in mathematics and science: the Average, or more specifically, the Arithmetic Mean. We hope you find it useful.

The Incremental Mean

One way to think about Q-learning is to think of it as simply *a method of computing averages*. Well, there's a bit more to it than that, but at its core, we are simply computing averages. So, let's begin there, with the Arithmetic mean (aka average), which in its simplest form with two operands is:

$$Q = \frac{x1 + x2}{2}$$

where Q represents the mean. This can be generalized to any number of values as:

$$Q = \frac{1}{n}\sum_{i=1}^{n} x_i$$

From a computational perspective, there are two challenges with this equation. First, it requires a lot of storage to perform the calculation. Specifically, it requires at least *n* memory units. This is okay if *n* is small, but becomes problematic if *n* is sufficiently large.

Second, this equation is prone to another problem called overflow. To illustrate this point, let's compute the average of 6 and 8, which is 7. Only using your hands, hold up six fingers. Now add 8 fingers to it? You can't do it (at least if you are like most people who are limited to 10 fingers between their two hands). And this is a problem because, while we can represent 6, 8, and the answer 7 on our fingers, we can't compute the answer using this method of summing everything first. The same problem applies to numeric representations. We can have two 64-bit integers (for example) where their average is also a 64-bit integer. However since this method first requires summing, the intermediate result can be greater than 64-bits, resulting in an overflow.

Okay, all of that is simply to say, we need a more efficient way of computing the mean. And we have one. The "trick" is to realize that:

$$\frac{x_1}{2} = x_1 - \frac{x_1}{2}$$

which allows us to rewrite the mean equation as:

$$Q = x_1 - \frac{x_1}{2} + \frac{x_2}{2}$$

This equation solves the overflow problem (at least for two operands).

When we assume i > 1 and

$$Q_1 = x_1$$

using induction, we can generalize this equation for *n* operands as:

$$Q_i = Q_{i-1} + \frac{1}{i}(x_i - Q_{i-1})$$

where we iteratively use this equation beginning with *i=2 (since Q_1 is given)* and incrementing until *i=n*.

Note: This equation is formally derived in Chapter 2.4 of Sutton's Reinforcement Learning book and by David Silver in his online RL course video beginning at the 26:20 mark.

Now, not only have we solved the overflow problem, but we have also addressed the storage problem because we never hold all *n* elements in memory at once. We simply iterate through this equation in a for loop, one element at a time, and compute the mean "incrementally". We hold the last mean and we hold the new value; that's all. *It is the "incremental" computation of the mean that is core to Q-learning*.

As mentioned above, this derivation is shown (with slightly different notation) in section 2.4 of Sutton's Reinforcement Learning book. Sutton generalizes the above equation as:

$$NewMean \leftarrow OldMean + StepSize\,[newX_i - OldMean]$$

A "modified" Incremental Mean as the Q-Learning Update Rule

Now that we understand the form of the Incremental Mean equation, let's rewrite the equation using our variables:

$$Q' = Q + \alpha\,(newX_i - Q)$$

where Q' is the NewMean, Q is the OldMean, and alpha is StepSize (which is a generalized version of 1/i). This use of alpha, which does not need to equal 1/i, is why I call this the *modified* incremental mean. Alpha is called the learning rate. Here's why: In the Q-Learning algorithm, we can adjust this value, alpha, in ways that place more emphasis on the new values of X_i or more emphasis on the previous means. Notice how alpha biases whether to emphasize learning new values or remembering old means. Examining the extremes; if alpha is 0,

$$Q' = Q$$

which means that we are biased toward the previous means (or the historical values), which will always be Q_1. When alpha is 1,

$$Q' = newX_i$$

which means we are biased toward the new value and we will not remember anything from the past. Generally, alpha will fall between 0 and 1.

Now, let's generalize this equation further. Instead of computing one mean in this manner, let's compute several means using this modified incremental mean equation. To do this, we could make one large one-dimensional array. Alternatively, we're going to simply use a two-dimensional array, which we'll continue to call Q, that we will index by *s* and *a*, resulting in the equation:

$$Q'\left(s,a\right) = Q\left(s,a\right) + \alpha\left(newX_i - Q\left(s,a\right)\right)$$

Indexing Q (or Q') with *s* and *a* simply points to a specific mean in the array. This use of Q' is simply notational and differs ever-so-slightly from Sutton's notation. But I'm hoping that this notation adds clarity to this derivation: Just remember that in this case, after the Q' update there is an implied:

$$Q \leftarrow Q'$$

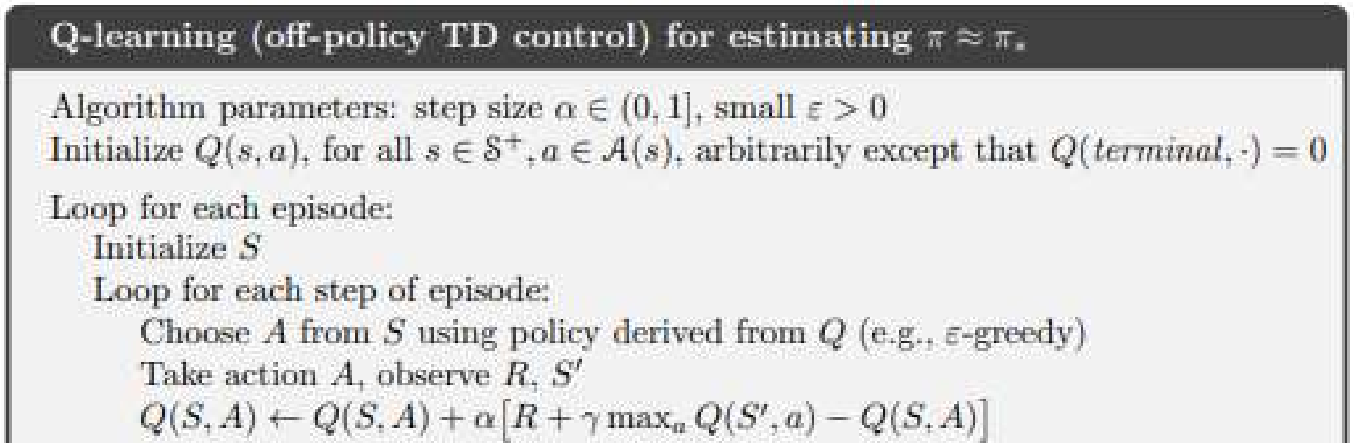
Okay, now all that's left now is to discuss is newX_i. I'll try this conceptually. Imagine that we have moved from state s to state s'. Two things need to happen. First, we need to get any reward, *r*, (a zero, a positive, or a negative value) for moving to this state s'. Second, we have to get any discounted value from taking the best action, *a'*, from that state *s'*. Mathematically, this is:

$$newX_i = r + \gamma Max_{a'}\left(s', a'\right)$$

Now we replace newX_i and arrive at:

$$Q'\left(s,a\right) = Q\left(s,a\right) + \alpha\left(r + \gamma Max_{a'}\left(s', a'\right) - Q\left(s,a\right)\right)$$

Some minor notational differences aside, this matches what is given in the Q-Learning algorithm below for the Q-Learner update step (from Chapter 6.5 of Sutton's RL book):



```
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
```

```
    $S \leftarrow S'$
  until $S$ is terminal
```

Rewriting the Q-Learning Update Rule

The equation can be rewritten equivalently by collecting the Q(s, a) terms together, as:

$$Q'\left(s,a\right)=\left(1-\alpha\right)Q\left(s,a\right)+\alpha\left(r+\gamma Max_{a'}Q\left(s',a'\right)\right)$$

Also notice the equivalence of:

$$Max_{a'}Q\left(s',a'\right)\equiv Q\left(s',ArgMax_{a'}Q\left(s',a'\right)\right)$$

which enables the equation to be rewritten equivalently as:

$$Q'\left(s,a\right)=\left(1-\alpha\right)Q\left(s,a\right)+\alpha\left(r+\gamma Q\left(s',ArgMax_{a'}Q\left(s',a'\right)\right)\right)$$

This is the form of the update rule Dr. Balch uses in the video lessons.

Example

Let's walk through a conceptual "learning" example. Let's say that my wife wants to time the completion of her Yoga workout to coincide with when I arrive home after my evening commute. She keeps a spreadsheet of my arrival times. On day one, I get home at 6:15. So, she can time her completion for 6:15. On day two, I get home at 5:55. So, she will now time her completion at 6:05, which is the mean of the two values. Let's say that this goes on for a month, where my wife might "learn" that my arrival time is, on average, 6:08. Notice how she was able to use data daily and did not have to wait for a month before performing her calculation. To make this example multi-dimensional, we can imagine that my wife keeps three means: one when I drive, one when I take the bus, and one when I take the ferry. To make this two-dimensional, we can add the *day of the week* as a second dimension. So, her Q table might be Q[day_of_the_week, transportation_mode].

Essentially, this is similar to how the update rule works with our Q-learning agent. It learns with each new "sample" but is able to use what it has learned so far. And just as we saw our expectations get better with more samples in P1 and with more rolls of a die, the agent learns better information (and is, therefore, able to take better actions) with more samples.

Conclusion

The intent of this post was to take some of the mystery out of the Q-Learning Update rule equation. At its core, the equation is essentially computing the mean of many entries in a big two-dimensional table. We can think of "learning", in this context, as arriving at an "expectation" (a.k.a, mean). (Please recall "expectations" and "expected values" discussions from Project 1).

PDF

Q_Learner_Derivation_Summer2022.pdf