Introduction to Nash Equilibria: Friend or Foe Q-Learning

Making robots tip the scales

Austin Nguyen Apr 21, 2020 · 7 min read ★

Photo by Toa Heftiba on Unsplash

For whatever reason, humans innately possess the ability to collaborate. It's become so commonplace that its nuances slip right under our noses. How do we just *know* how to coordinate when moving a heavy couch? How do we *reason* splitting up in a grocery store to minimize time? How are we able to *observe* others' actions and *understand* how to best *respond*?

Here's an interpretation: we reach a balance. An **equilibrium.** Each person takes actions that not only best complements the others' but altogether achieves the task at hand most efficiently. This application of equilibria comes up pretty often in game theory and extends to multi-agent RL (MARL). In this article, we explore two algorithms, Nash Q-Learning and Friend or Foe Q-Learning, both of which attempt to find multi-agent policies fulfilling this idea of "balance." We assume basic knowledge of single-agent formulations and Q-learning. For more background on MARL

and basic principles specifically pertaining to it, check out my previous article:

Photo by Erik Mclean on Unsplash

What Makes an Optimal Policy...Optimal?

Multi-agent learning environments are typically represented by Stochastic Games. Each agent aims to find a policy that maximizes their own expected discounted reward. Together, the overall goal is to find a **joint** policy that gathers the most reward for **each agent**. This joint reward is defined below in the form of a value function:

$$v^i(s, \pi^1, \pi^2, \ldots, \pi^n) = \sum_{t=0}^{\infty} \beta^t E(r_t^1 | \pi^1, \pi^2, \ldots, \pi^n, s_0 = s).$$

This goal applies to both *competitive* and *collaborative* situations. Agents can find policies that best *counter* or *complement* others. We call this optimal policy the **Nash Equilibrium.** More formally, it is a policy such that has this property:

$$v^i(s, \pi_*^1, \ldots, \pi_*^n) \geq v^i(s, \pi_*^1, \ldots, \pi_*^{i-1}, \pi^i, \pi_*^{i+1}, \ldots, \pi_*^n) \quad \text{for all } \pi^i \in \Pi^i,$$

At first, it seems like we're beating a dead horse. The best policy gathers the most reward, so what?

Underneath all the fancy greek letters and notation, the Nash Equilibrium tells us a bit more. It says that each agent's policy in Nash Equilibrium is the **best response to the other agents' optimal policies.** No agent is incentivized to change their policy because any tweak gives less reward. In other words, all of the agents are at a standstill. Landlocked. Kind of trapped in a sense.

Photo by NeONBRAND on Unsplash

To give an example, imagine a *competitive* game between two small robots: C3PO and Wall-E. During each round, they each choose a number one through ten, and whoever selects the higher number wins. As expected, both pick the number ten every time as neither robot wants to risk losing. If C3PO were to choose any other number, he would risk losing against Wall-E's optimal policy of always choosing ten and vice versa. In other words, the two are at an equilibrium.

Nash Q-Learning

As a result, we define a term called the **Nash Q-Value:**

$$Q_*^i(s, a^1, \ldots, a^n) = r^i(s, a^1, \ldots, a^n) + \beta \sum_{s' \in S} p(s'|s, a^1, \ldots, a^n) v^i(s', \pi_*^1, \ldots, \pi_*^n),$$

Very similar to its single-agent counterpart, the Nash Q-Value represents an agent's expected future cumulative reward when, after choosing a specific

joint action, *all agents follow a joint Nash Equilibrium policy*. It can be viewed as a state-action pair's "best-case scenario" reward gain. This definition lends itself to the multi-agent version of Q-learning: Nash Q-Learning.

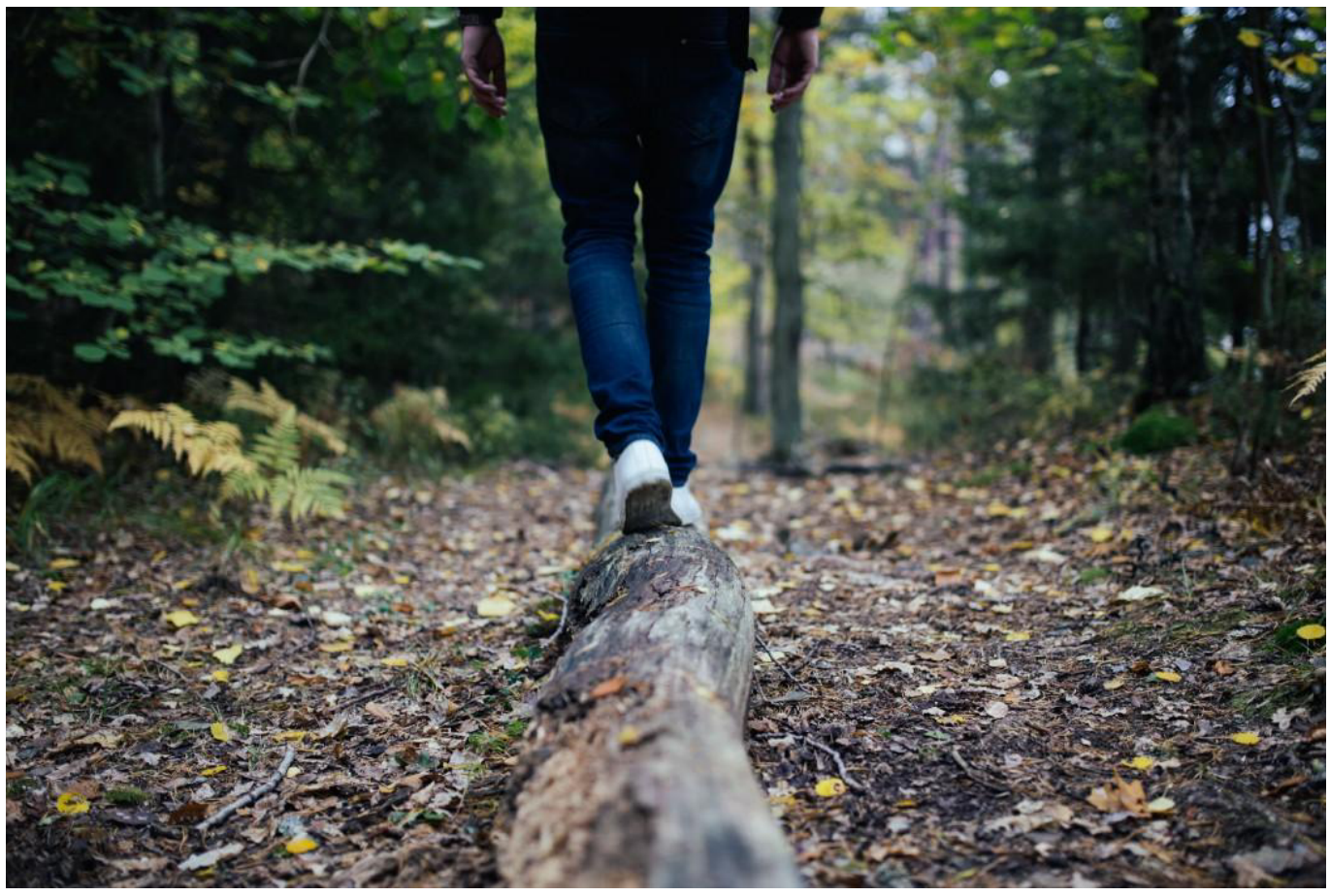

Photo by Jon Flobrant on Unsplash

In single-agent Q-learning, we update Q values using the temporal difference (TD) with the equation:

$$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \max_{a'} Q(a',s') - Q(a,s) \right)$$

where gamma represents the discount factor and alpha is the learning rate. Here's the slightly boring part: Nash Q-learning reduces to Q-learning. More explicitly, the update equation for Nash Q-Learning is:

$$Q^i_{t+1}(s,a^1,\ldots,a^n) = (1-\alpha_t)Q^i_t(s,a^1,\ldots,a^n) + \alpha_t \left[r^i_t + \beta Nash Q^i_t(s') \right]$$

where we compute the Nash Q-value by maximizing over the action space. Switch around a few terms and you arrive at a nearly identical update equation, the only difference being a joint action space. So, what was the point of all this?

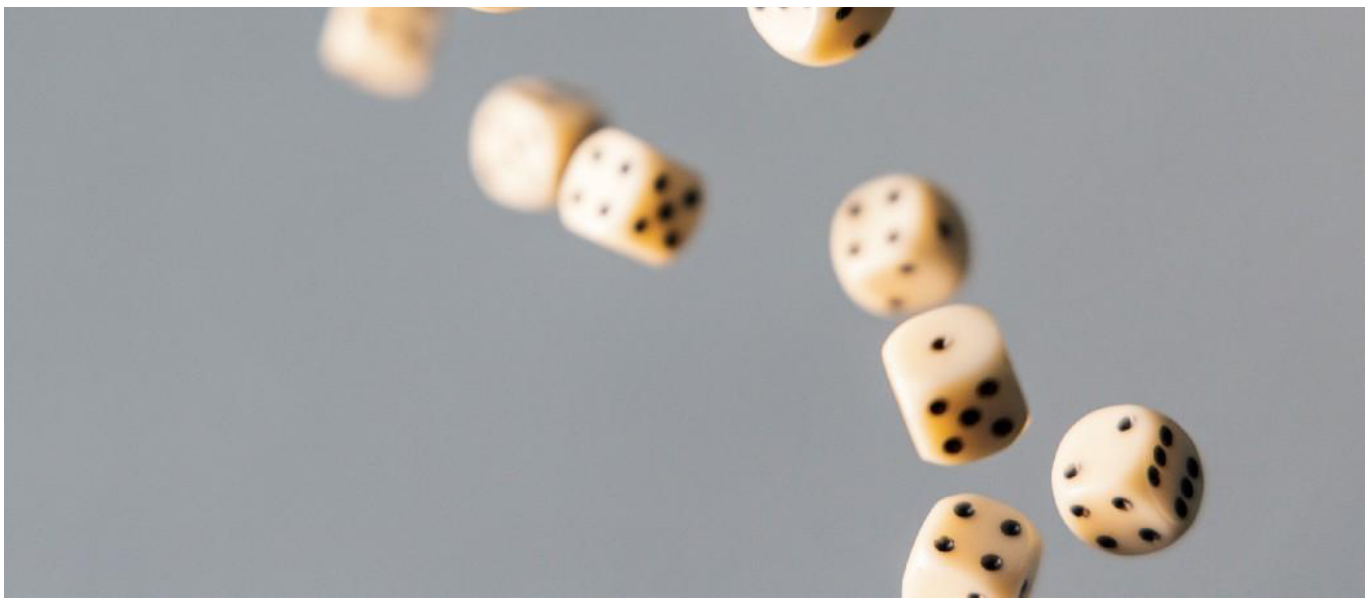

Photo by Riho Kroll on Unsplash

The point: we can't always assume that single-agent logic applies to multi-agent settings. In other words, algorithms that apply to MDPs (single-agent RL) aren't always translatable to Stochastic Games (multi-agent RL). Nash Q-Learning just happens to be a somewhat special case. Hu and Wellman [1], among many other things in the paper, prove that Nash Q-Learning always converged. The fact that its update equation looks identical to Q-learning is intuitive but not always assumable beforehand.

The entire Nash Q-learning algorithm is analogous to single-agent Q-learning and is shown below:

Initialize:
Let $t = 0$, get the initial state s_0.
Let the learning agent be indexed by i.
For all $s \in S$ and $a^j \in A^j$, $j = 1, \ldots, n$, let $Q_t^j(s, a^1, \ldots, a^n) = 0$.
Loop
Choose action a_t^i.
Observe $r_t^1, \ldots, r_t^n; a_t^1, \ldots, a_t^n$, and $s_{t+1} = s'$
Update Q_t^j for $j = 1, \ldots, n$
$Q_{t+1}^j(s, a^1, \ldots, a^n) = (1 - \alpha_t) Q_t^j(s, a^1, \ldots, a^n) + \alpha_t [r_t^j + \beta NashQ_t^j(s')]$
where $\alpha_t \in (0, 1)$ is the learning rate, and $NashQ_t^k(s')$ is defined in (7)
Let $t := t + 1$.

Friend or Foe Q-Learning

Q-values have a natural interpretation. They represent a state-action pair's expected cumulative discounted reward, but how does that motivate our update equation? Let's take a look at it again.

$$Q_{t+1}^i(s, a^1, \ldots, a^n) = (1 - \alpha_t) Q_t^i(s, a^1, \ldots, a^n) + \alpha_t \left[r_t^i + \beta NashQ_t^i(s') \right]$$

This is a weighted sum where alpha is the learning rate, a value in the interval (0,1). The first term is our Q value at a previous time step

multiplied by some fraction. Let's dissect the second term, our learning rate multiplied by a sum of two other terms.

The first part of the second term is a reward we get after executing an action while the other is a discounted, Nash Q-value maximized over actions for the next state. Remember, this Nash Q-value equals the expected, best-case, cumulative reward after a given state. Putting the two parts together, we have a **sum of expected rewards** where we assume the *best-case scenario* at the next state**.** It's another definition for the current Q value, just an updated version! More formally, this sum represents a **closer estimate of the actual Q value.** Zooming out and looking at the entire update equation, we can interpret it as "nudging" the current Q-value closer to this better estimate.

Photo by Hudson Hintze on Unsplash

In Friend or Foe Q-Learning (FFQ), Littman [2] uses this interpretation to create an intuitive algorithm for multi-agent systems. FFQ starts by letting each agent maintain their *own* Q function, contrary to Nash Q-learning. We assume that agents are aware of others' action spaces but don't know what actions others will choose at any given moment.

Also, Littman generalizes to systems where there is a combination of collaborators and adversaries, hence the name Friend or Foe. Let's use C3PO (agent 1) and Wall-E (agent 2) as examples again, and view things from C3PO's point of view. If the two are working together, C3PO's Nash-Q value would be replaced with:

$$\text{Nash}_1(s, Q_1, Q_2) = \max_{a_1 \in A_1, a_2 \in A_2} Q_1[s, a_1, a_2]$$

as expected, as both agents attempt to maximize their “stream” of rewards, similar to the Q update function from before. However, if the two robots are competing against each other, C3PO’s Nash-Q would be replaced with:

$$\text{Nash}_1(s, Q_1, Q_2) = \max_{\pi \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} \pi(a_1) Q_1[s, a_1, a_2]$$

Notice how we first minimize over Wall-E’s actions before we maximize over C3PO’s actions. In other words, while Wall-E attempts to minimize C3PO’s stream of rewards, C3PO learns how to **best respond** to Wall-E’s counteraction, updating his Q-value accordingly. This is an example of when the idea of Nash Equilibria shows its colors: agents learning the optimal way to complement each other.

Photo by Pavan Trikutam on Unsplash

More generally, this can be adapted to larger systems of agents. The Nash-Q value becomes as follows:

$$\text{Nash}_i(s, Q_1, \ldots, Q_n) =$$

$$\max_{\pi \in \Pi(X_1 \times \cdots \times X_k)} \min_{y_1, \ldots, y_l \in Y_1 \times \cdots \times Y_l} \sum_{x_1, \ldots, x_k \in X_1 \times \cdots \times X_k}$$

$$\pi(x_1) \cdots \pi(x_k) Q_i[s, x_1, \ldots, x_k, y_1, \ldots, y_l].$$

where X represents the set of all collaborators while Y represents the set of all adversaries. Similar to Nash Q-learning, Littman[2] showed that Friend or Foe Q-learning also converged. By replacing these Nash Q-values appropriately, a *distributed* system of agents can learn a policy that reaches equilibrium, if one exists.

First Steps

These algorithms are what some consider the “first steps” into the multi-agent learning scene. While the ideas are relatively simple, the results are powerful. Next time, we’ll look into more novel, modern-day approaches to solving complex, multi-agent environments! For a more in-depth look at Nash Q-learning and Friend or Foe Q-learning, feel free to take a look at the two papers listed below.

References

[1] J. Hu and M. Wellman, Nash Q-Learning for General-Sum Stochastic Games (2003), Journal of Machine Learning Research 4

[2] M. Littman, Friend-or-Foe Q-learning in General-Sum Games (2001), ICML

> *From the classic to state-of-the-art, here are related articles discussing both multi-agent and single-agent reinforcement learning:*