01_Dyna

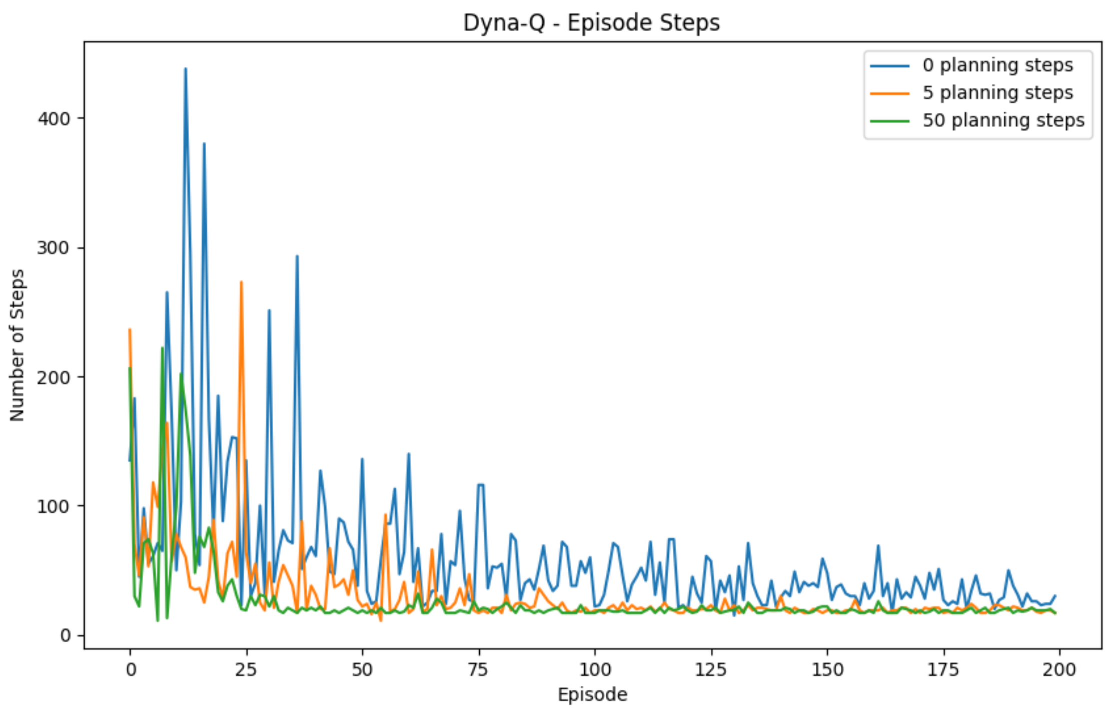
The first model-based RL paper I explored was Rich Sutton's *"Dyna: An Integrated Architecture for Learning, Planning, and Reacting"*. Link to the paper. Dyna cleverly combines model-based and model-free RL in a unified framework.

Dyna Architecture

Dyna operates under a deterministic environment assumption. We have lookup tables for both our models and Q-values. Think of it as standard Q-learning, but with a twist: rather than just updating Q-values from direct interaction with the environment, Dyna stores these interactions in a model (like a hash map) for later use. The agent then samples from this model for *n* planning steps, refining the Q-values based on simulated experiences *('inside imagination')*

This approach means less real-world interaction and more computation in the agent's internal model—ideal if you have some GPU muscle to spare. The reward? Greater sample efficiency.

Here’s a comparison between Q-learning and Dyna with 5 and 50 planning steps in a simple Gridworld. The Y-axis shows the steps until episode termination, while the X-axis tracks episodes. Notice how Dyna with just 5 planning steps reaches optimal performance by episode 100, while with 50 planning steps, it converges around episode 30. In contrast, vanilla Q-learning fluctuates until about 200 episodes.



Limitations

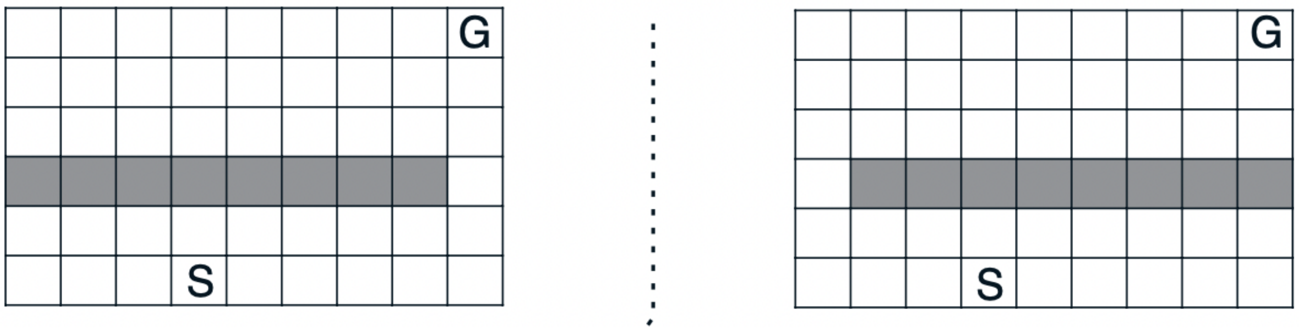
While Dyna is great, it’s not without its flaws. It's a framework, more of a rough sketch for integrating planning into direct RL rather than a full-blown algorithm. Dyna-Q, the example algorithm, highlights a few key limitations:

1. Stochastic Environments

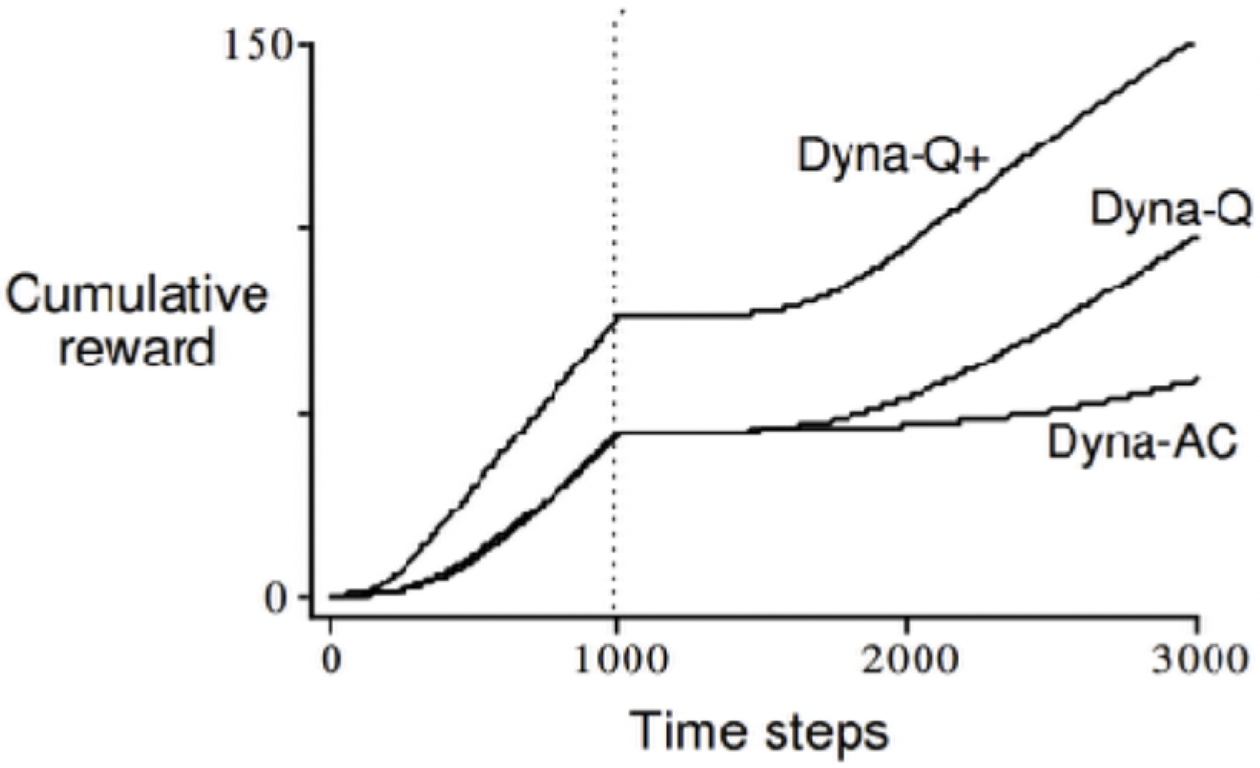
Dyna assumes determinism, which isn’t realistic for most real-world scenarios. If the environment changes unexpectedly, our model becomes inaccurate, resulting in poor updates and policies.

For example, consider a Gridworld with a mid-level partition wall. Dyna-Q can navigate this easily, but if we shift the wall after 1000 steps, things get tricky:

Left: Original environment Right: Changed environment



Dyna-Q struggles in this altered scenario:



Dyna-Q+ addresses this by tracking when state-action pairs were last visited, adding a bonus for transitions not visited recently. This incentivizes exploration: $R + k\sqrt{\tau}$

However, it still assumes a deterministic world. Today, we have tools like Gaussian Processes and probabilistic MLPs to model stochastic environments, which would be a natural extension for this.

2. Tabular Limitations

Dyna is tabular—Q-values and models are lookup tables, and states/actions are discrete. No generalization. To move beyond tabular, we need function approximators like linear models or deep networks (think DQN). Of course, these come with challenges, like potential convergence issues, but they expand Dyna's capabilities beyond simple grids.

In fact, the "Replay Buffer" from DQN feels like an evolution of Dyna's hash-map model, storing experiences for future Q-updates. "Replay Buffer" that is introduced in DQN paper seems to serve the same purpose as the hashmap-based model that Dyna uses to store experiences and sample those experiences to do Q-learning updates.

It's incredible to see how these foundational ideas from 1991 have evolved and influenced modern RL techniques. Sutton's vision was way ahead of its time.