Training Mario Experiments Journal

Playing Atari with Deep Reinforcement Learning

Key contributions: Introduced deep Q-networks (DQN) that combine reinforcement learning with deep neural networks, enabling agents to learn directly from high-dimensional sensory input and achieve human-level performance on Atari games using **experience replay** and **target networks**.

In smaller environments one can compute the Q-function ($Q(s, a)$) by tabulating experiences. But for environments with a large number of states to become computationally tractable, like Atari games, this paper successfully leverages neural networks to approximate the Q-function.

One issue that arises when bootstrapping the Q-function with itself, **and** using it for choosing actions is that the agent ends up "going down the rabbit hole" and usually results in divergent or sample inefficient training. For this reason we perform Q-value updates using:

$Q(s, a) \leftarrow r + \gamma * max_{a'}(Q_{target}(s', a'))$

Where $Q_{target} \leftarrow Q$ every K steps.

Additional tricks/notes

Training instability: I've taken two additional steps to stabilize training: * L1+MSE loss function * Gradient clipping

Previous action as input: The agent quickly learns that run and jump buttons together provide a lot of reward, but Mario only jumps if you've not pressed that button before. This in turns makes Mario continuously run into the mushroom enemies. To distinguish the states where pressing jump will result in a jump vs. continue because we have already pressed I've added the previous action as input. It does improve this behavior slightly.

Softmax action selection: Choosing highest Q-value action at every step results in Mario getting stuck in parts where the agent is pressing against an object. To perform more randomized actions I've implemented softmax action selection, which gretly improves agent performance by encouraging more exploration (i.e. in between using ϵ-greedy and max operator for selecting actions).

Comparison between a few agents

Agent	Action Selection Method	Epsilon Schedule	Softmax Temperature	Characteristics
Random Agent	ϵ-greedy	$\epsilon = 1$	N/A	Purely random actions

Agent	Action Selection Method	Epsilon Schedule	Softmax Tempera-ture	Characteristics
Max Agent	ϵ-greedy	$1 \rightarrow 0.02$	N/A	Greedy with decaying exploration
Softmax Agent 0.1	ϵ-softmax	$1 \rightarrow 0.02$	0.1	More deterministic, favors highest Q-values
Softmax Agent 0.2	ϵ-softmax	$1 \rightarrow 0.02$	0.2	More exploration, less deterministic than 0.1

Prioritized Experience Replay

Key contribution: This paper introduces a method to prioritize replayed experiences by sampling more important transitions more frequently, based on the magnitude of their temporal-difference (TD) error. This improves learning efficiency and accelerates convergence by focusing updates on experiences that have the greatest impact on learning.

The probability of picking experience i changes from,

$$P(i) = \frac{1}{N}$$

to,

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

where p_i is the priority of transition i and α determines how much prioritization is used ($\alpha = 0$ corresponds to uniform sampling). This priority can either be rank-based or simply the TD-error.

Additional tricks/notes

Weighted importance sampling: Since we are changing the i.i.d. nature of uniform sampling of experiences, we will introduce a bias that will need to be corrected. See details in Weighted importance sampling for off-policy learning with linear function approximation. This involves multiplying the gradient updates by,

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

where β controls the amount of importance-sampling correction.

TODO Reading

- Noisy Networks for Exploration