Deep Reinforcement Learning

Dr.Girish Varma
Sharfuddin Mohammed

Recap of Model-free RL

Recap

- Learning from short term rewards and modifying your behaviour based on the observation from the environment to maximize long-term rewards
- We have also looked at how to model our environment and agent as an MDP
- **Value functions:** State value function V(s) and action value function Q(s,a)
- **Policy iteration** and **Policy evaluation** for evaluating and improving the policies
- **Value Iteration**:Find state value function and act greedily according to it,gives the state value function for the optimal policies
- The importance of value-functions and **Bellman-Optimality**
- **Model-based** approaches which assume a complete knowledge of the environment,and transition dynamics.

Recap:Model-Free Control

- For almost all real-world problems we have an incomplete knowledge of the nondeterministic nature of the environment
- How to find optimal policies and value functions in such cases?
- **Monte-Carlo** and **Temporal-Difference(TD)** methods for finding the optimal-policies
- Start with a random-policy,evaluate the policy and improve the policy by acting greedily(using soft-policies).This is called **on-policy**
- Act greedily with respect to the value-function **off-policy**

Recap:On-Policy Monte-Carlo Control

- **Policy Evaluation:** Monte-Carlo policy evaluation, $Q \approx q_\pi$,All state-action pairs are explored infinitely many times $\lim_{k \to \infty} N_k(s, a) = \infty$

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

- **Policy Improvement:** ϵ-greedy policy improvement,ϵ reduces to zero asymptotically. $\epsilon_k = \frac{1}{k}$

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Recap:On-Policy TD Control (SARSA)

- **Policy Evaluation:** Initialize Q(s,a) randomly and apply on-policy TD updates to Q-values

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

- **Policy Iteration:** same policy improvement as on-policy MC control

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

- The above procedure is for TD(0) SARSA

Recap:Q-Learning

- Allow for both behaviour and target policies to improve
- The target policy is greedy w.r.t $Q(s, a)$
- The Q-Learning target simplifies to

$$R_{t+1} + \gamma Q(S_{t+1}, A')$$
$$R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a'))$$
$$R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

Recap:Value Function-Approximation

- The infeasibility of maintaining a Q-table and storing for all state-action pairs
- Approximate Q-functions using a parametrized function (in most cases a Deep-Network)
- Learn the parameters of the value-function of the optimal-policy
- Store experience <s,r,s'> in the replay-buffer
- Draw samples randomly from the replay-buffer and use these to perform batch-training

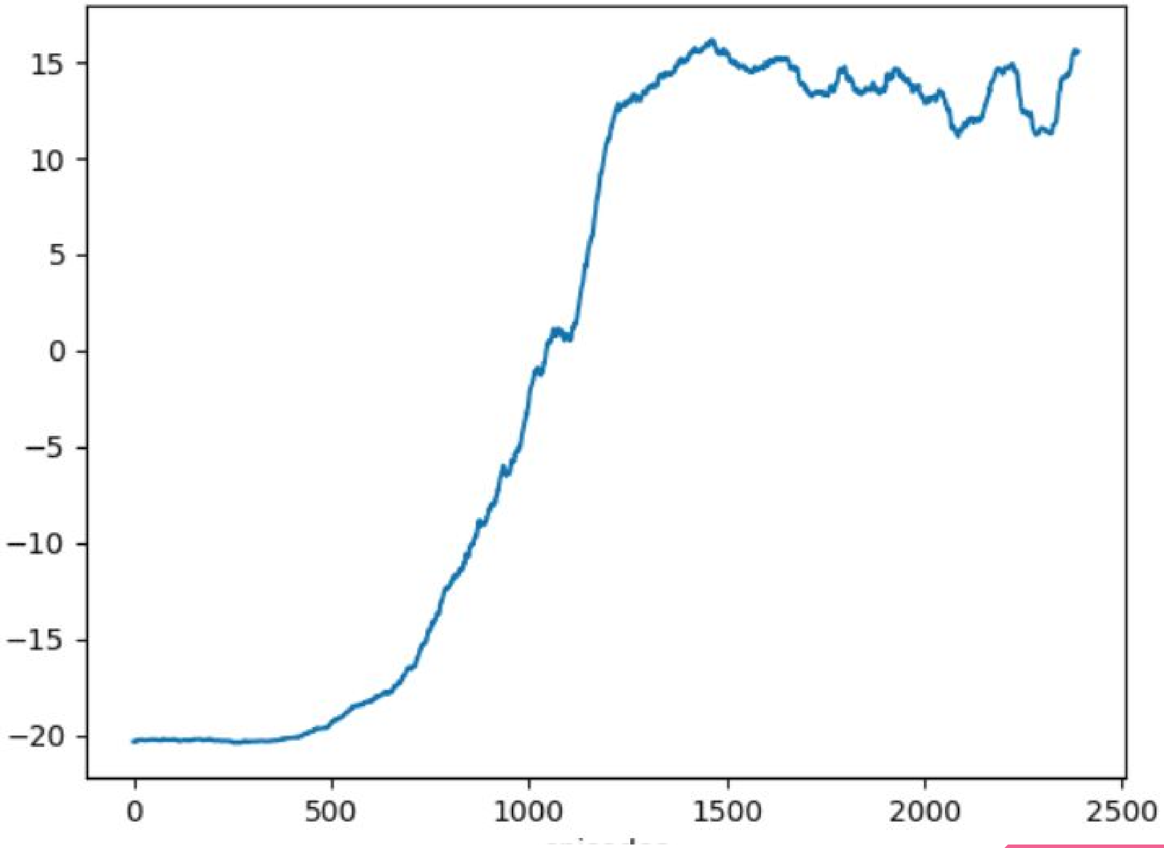
The DQN-Class of Algorithms

DQN-Breakout

- End-to-end learning of values Q(s, a) from pixels s
- Input state s is stack of raw pixels from last 4 frames.Each frame is of size (128x284)
- Output is Q(s, a) for 2 possible actions in the breakout environment
- Environment randomly samples {2,3,4} times take the same action
- Demo

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}}\right)^2\right]$$

Rewards-per episode on Breakout for DQN

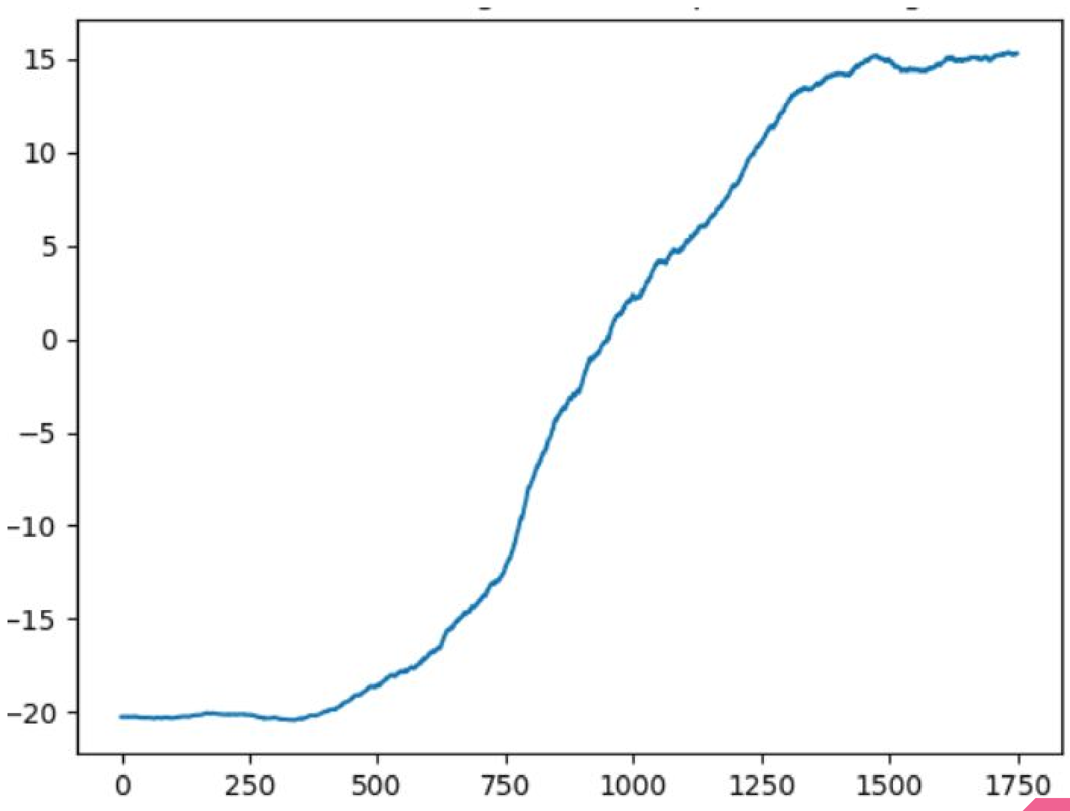


Double DQN-Breakout

- Train 2 instances of the same Q-network, Q1 and Q2
- Do Q-learning on both,but
  - never on the same time steps (Q1 and Q2 are independent)
  - pick Q1 or Q2 randomly with prob. 0.5 to be updated on each step
- If updating Q1 then use Q2 for value of the next state

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \\ + \alpha\Big(R_{t+1} + Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)\Big)$$

Reward-per Episode for Double DQN

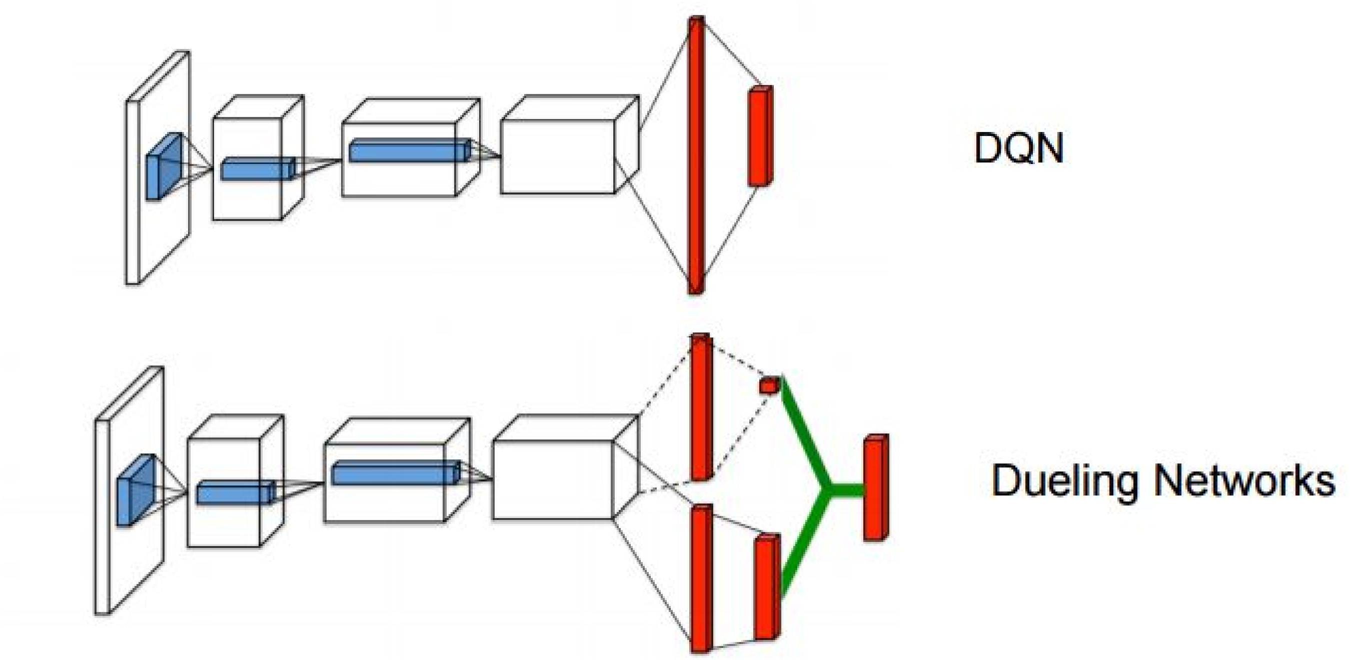


Dueling DQN-Breakout

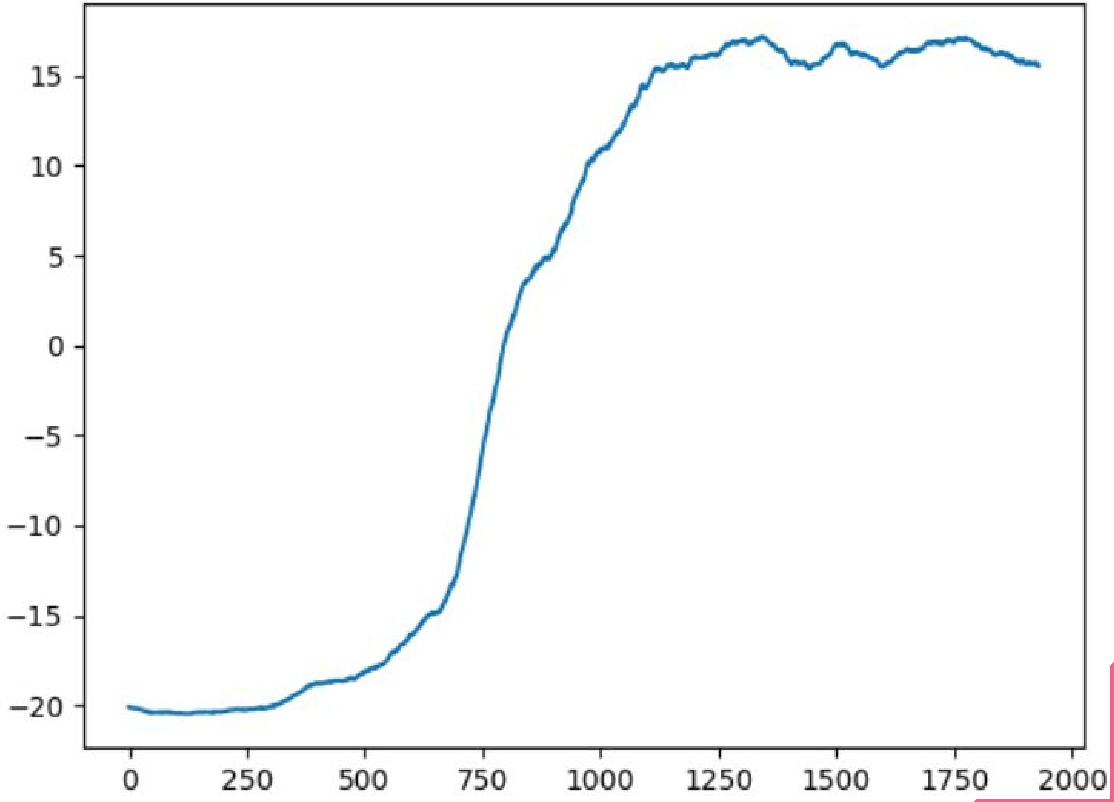
- Instead of using Q-network for making decisions,estimate how good an action-value is better than an estimate of the state-value function and make decisions based on it
- Split the Q-network into two channels
  - **Action-independent** value function V(s,v)
  - **Action-dependent** advantage function A(s, a, w)
- Advantage function is defined as follows

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s).$$

Dueling DQN-Breakout



Reward per Episode for Dueling-DQN



Policy-Gradients and Actor-Critic Methods

Policy gradient methods

- DQN algorithms were off-policy greedy algorithms
- Rather than learning value functions for optimal policy and then using them to find the optimal policy,start with a policy that is updated based on the value-functions
- We also parametrize the policy for a state and produce a probability vector for each action

$$\pi_\theta(s, a) = \mathbb{P}\left[a \mid s, \theta\right]$$

- The objective of all Reinforcement Learning algorithms can be formulated as maximizing the expected return from the start-state.$J(\theta) = E\pi_\theta[R_0]$

Monte-Carlo Policy Gradients and PG-Theorem

- Monte Carlo policy gradient methods apply direct gradient-based optimization to the reinforcement learning objective.

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)\gamma^t R_t] = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b(s_t))],$$

- Policy-gradient theorem generalizes the likelihood ratio and replaces instantaneous rewards with long-term Qπ (s, a)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^{\pi_\theta}(s,a)\right]$$

Monte-Carlo Policy Gradient(REINFORCE)

- Update parameters by stochastic-gradient-descent
- Use policy-gradient theorem
- Use return Vt as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

Actor-Critic Method

- Monte-Carlo policy gradients suffer from high variance
- Instead of using returns to estimate the Q-value we use a critic to estimate the value-functions

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain two sets of parameters
  - **Critic** Updates action-value function parameters w
  - **Actor** Updates policy parameters θ, in direction suggested by critic

Actor-Critic Method

- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)$$

- **Critic** updates w by linear TD(0)
- **Actor** updates θ by policy gradient

Advantage-Actor-Critic Method

- We subtract a baseline function B(s) from the policy gradient
- This can reduce variance, without changing expec $V^{\pi_\theta}(s)$
- A good baseline is the state value function B(s) =
- Rewrite the policy gradient using the advantage function

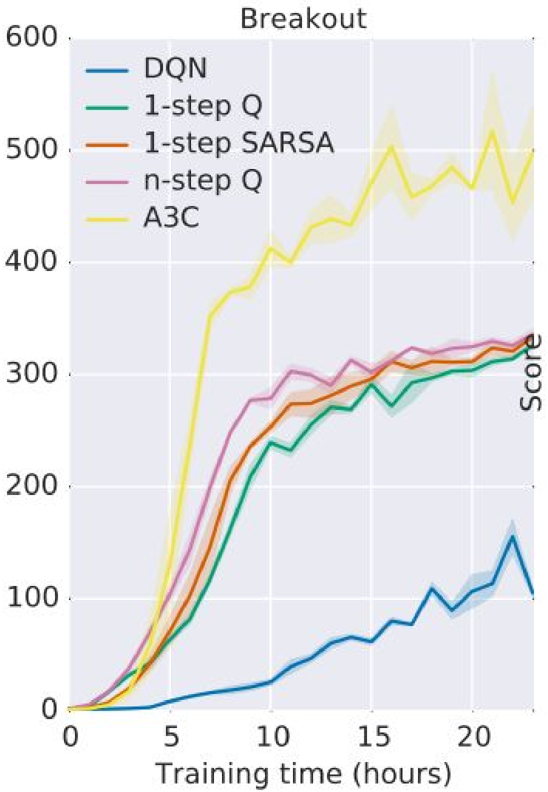
$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$

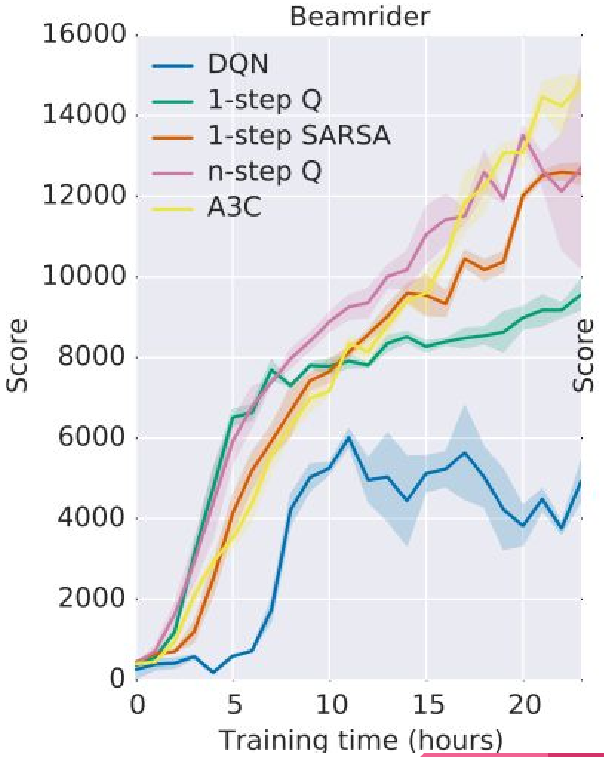
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; A^{\pi_\theta}(s,a)\right]$$

Asynchronous-Advantage-Actor-Critic(A3C)

- Algorithm that is faster, simpler, more robust, and is able to achieve much better scores on the standard battery of Deep RL tasks
- Unlike DQN, where a single agent represented by a single neural network interacts with a single environment, A3C utilizes multiple incarnations of the agent to learn more efficiently
- There is a global network and multiple worker agents which each have their own set of network parameters
- Each of the agents interacts with it’s own copy of the environment at the same time as the other agents are interacting with their environments

Performance of A3C (Adapted from Silver et.Al,2016)





Demo On BeamRider and Seaquest

Motivation and Current-Research in Deep-RL

My current Work in Deep-RL

- Solving problems with partial observability and irregular reward intervals (eg.Tangram and maze-like games)
  - Using Recurrent-Neural nets for training agents
  - Using Q-prop and TRPO
- Applications to robotics in Planning and control (better Visual servoing)
  - Fitted Q-value iteration

Alpha GO and Alpha GO0

Why I want to do further research in Deep-RL?

Fascinated by the similarity of learning in these agents to human learning

A novel way of learning

Impact across different fields especially in Computer Vision and Robotics