Learning to Play in a Day: Faster Deep Reinforcement Learning by Optimality Tightening

Frank S. He[1,2] Yang Liu[1] Alexander G. Schwing[1] Jian Peng[1]

[1]University of Illinois at Urbana-Champaign [2]Zhejiang University

Motivation

Task: Deep Reinforcement Learning

- Given information and signals from environment
- Learn policies to maximize cumulative rewards

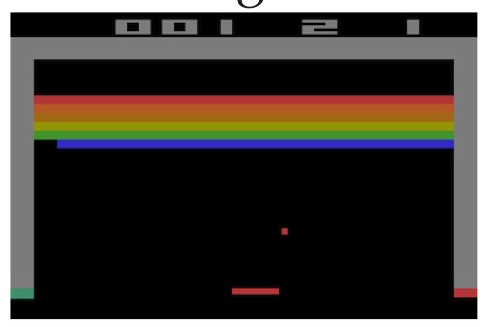
Environment is often complex and its output is of high dimensions

Contributions:

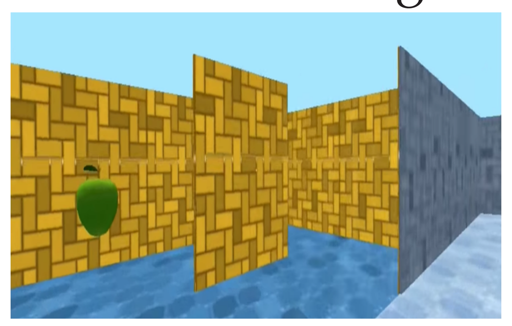
- How to reduce the computational resources?
- How to make learning process converge faster?
- How to obtain fast reward propagation?

Introduction

Atari games



Maze walking



Autonomous driving

Reinforcement learning:
Consider an agent operating over time $t \in \{1, \ldots, T\}$. At time t the agent is at an environment state s_t and reacts upon it by choosing action $a_t \in \mathcal{A}$. The agent will then observe a new state s_{t+1} and receive a numerical reward $r_t \in \mathcal{R}$. The goal of a reinforcement learning agent is to collect as much reward as possible.

Deep Q-learning: Given a state s, for each action $a \in \mathcal{A}$, function approximation $Q(s, a)$ estimates the expected cumulative future reward.

The core idea of Q-learning is the use of the Bellman equation

$$Q^*(s_t, a) = \mathbb{E}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

- Experience replay $\mathcal{D} = \{(s_j, a_j, r_j, s_{j+1})\}$ contains state-action-reward-future state-tuples drawn from game episodes.
- Fixed Q-function Q_{θ^-} is adopted for target $y_j = r_j + \gamma \max_a Q_{\theta^-}(s_{j+1}, a)$. Q_{θ^-} is updated by current Q_θ periodically. The dependence of the target on the parameter θ is ignored.

Q-function is updated by optimizing the following cost function with respect to the parameters θ via stochastic gradient descent:

$$\min_\theta \sum_{(s_j, a_j, r_j, s_{j+1}) \in \mathcal{B}} (Q_\theta(s_j, a_j) - y_j)^2$$

Policy: ϵ-greedy strategy makes the agent select actions randomly with probability ϵ or by following the strategy $\arg\max_a Q_\theta(s_t, a)$

Optimality Tightening

Bellman optimality equation:

$$Q^*(s_j, a_j) = r_j + \gamma \max_a Q^*(s_{j+1}, a) = \ldots$$

$$= r_j + \gamma \max_a \left[r_{j+1} + \gamma \max_{a'} \left[r_{j+2} + \gamma \max_{\tilde{a}} Q^*(s_{j+3}, \tilde{a}) \right] \right].$$

Question:

How to evaluate such a sequence?

Approach:

Take advantage of the episodes in the replay memory $\mathcal{D}$ and use samples (s_j, a_j, r_j, s_{j+1}) to evaluate the following inequalities:

$$Q^*(s_j, a_j) \geq \sum_{i=0}^{k} \gamma^i r_{j+i} + \gamma^{k+1} \max_a Q^*(s_{j+k+1}, a) = L^*_{j,k},$$

- $L^*_{j,k}$ is the lower bound of sample j and time horizon is k

$$Q^*(s_{j-k-1}, a_{j-k-1}) \geq \sum_{i=0}^{k} \gamma^i r_{j-k-1+i} + \gamma^{k+1} Q^*(s_j, a_j),$$

$$U^*_{j,k} = \gamma^{-k-1} Q^*(s_{j-k-1}, a_{j-k-1}) - \sum_{i=0}^{k} \gamma^{i-k-1} r_{j-k-1+i} \geq Q^*(s_j, a_j).$$

- $U^*_{j,k}$ is the upper bound of sample j and time horizon is k

Objective:

$$\min_\theta \sum_{(s_j, a_j, s_{j+1}, r_j) \in \mathcal{B}} (Q_\theta(s_j, a_j) - y_j)^2$$

$$\text{s.t.} \begin{cases} Q_\theta(s_j, a_j) \geq L^{\max}_j = \max_{k \in \{1, \ldots, K\}} L_{j,k} & \forall (s_j, a_j) \in \mathcal{B} \\ Q_\theta(s_j, a_j) \leq U^{\min}_j = \min_{k \in \{1, \ldots, K\}} U_{j,k} & \forall (s_j, a_j) \in \mathcal{B} \end{cases}$$

Optimization: RMSProp for stochastic gradient descent

$$\min_\theta \sum_{(s_j, a_j, r_j, s_{j+1}) \in \mathcal{B}} \Big[(Q_\theta(s_j, a_j) - y_j)^2 + \lambda (L^{\max}_j - Q_\theta(s_j, a_j))^2_+ + \lambda (Q_\theta(s_j, a_j) - U^{\min}_j)^2_+ \Big]$$

- $(x)_+ = \max(0, x)$ is the rectifier. λ is a penalty can be set as a large positive value or adjusted in an annealing scheme.
- The derivatives of $Q(s_j, a_j)$ not only depend on the Q-function from the immediately successive time step $Q(s_{j+1}, a)$ stored in the experience replay memory, but also on more distant time instances if constraints are violated

Computational Efficiency:

- Add real discounted return $R_j = \sum_{\tau=j}^{T} \gamma^{\tau-j} r_\tau$ to experience replay ($(s_j, a_j, r_j, R_j, s_{j+1}) \in \mathcal{D}$), thus computing discounted return over multiple time steps is in $O(1)$ time.
- R_j is also incorporated as an n-step lower bound to further stabilize the training.
- Set K small so that a minibatch of both bounds $U_{j,k}, L_{j,k}$ and targets y_j can be computed in a single GPU forward pass.

Implementation details

Gradient descent:

- Bounds and targets are evaluated by fixed target Q-function.
- Gradients are rescaled so that their magnitudes are comparable with or without penalty.

Sampling strategy: two constraint sampling strategies are tested

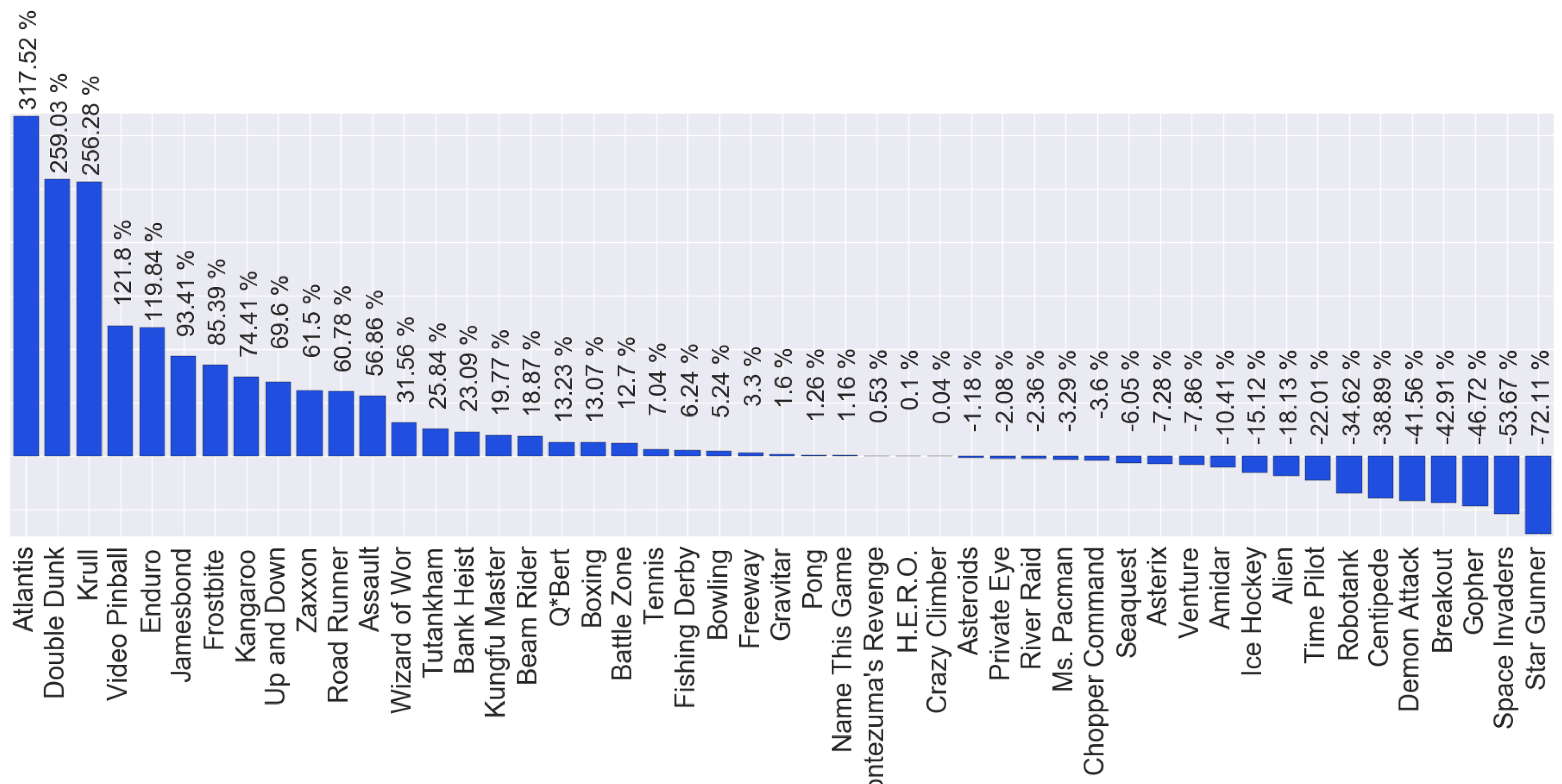
- K immediate predecessors and successors are selected as upper and lower bounds.
- Randomly choosing K out of 12 close instances as bounds for both directions.

Results

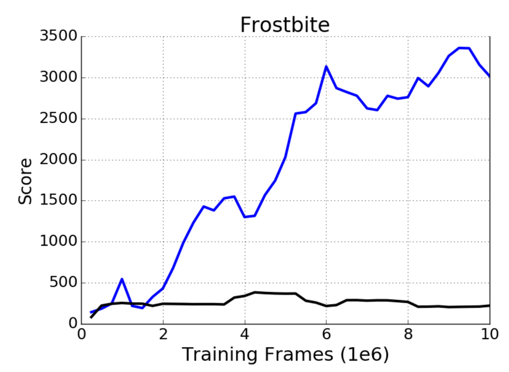
Experiment setup:

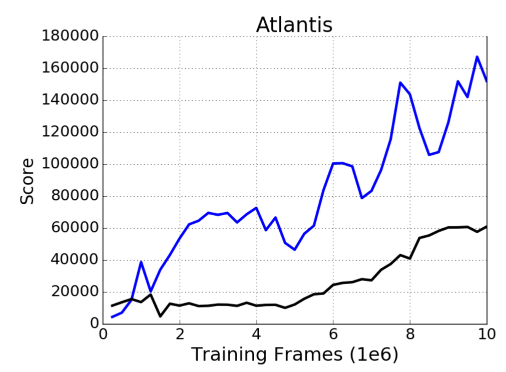
- $K = 4$ and $\lambda = 4$ are used.
- Identical network structure and hyperparameters as DQN.
- 30 no-op evaluation is used.

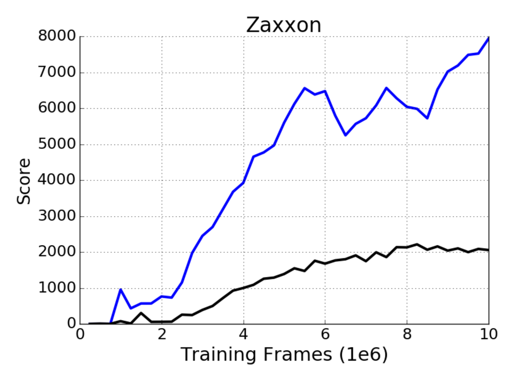
Improvements of our method trained on 10M frames compared to results of 200M frame DQN training:



Game scores for our algorithm (blue) and DQN (black) using 10M training frames (30 no-op evaluation/moving average over 4 points):







	Training Time	Mean	Median
Ours (10M)	**less than 1 day (1 GPU)**	**345.70%**	**105.74%**
DQN (200M)	more than 10 days (1 GPU)	241.06%	93.52%
D-DQN (200M)	more than 10 days (1 GPU)	330.3%	114.7%