Off-Policy Learning (Part 1)

Safe and Efficient Off-Policy Reinforcement Learning

The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning
Gruslys, A., Azar, M.G., Bellemare, M.G. and Munos, R., ICLR 2018

HaoChih, Lin
2019-April-09
Safe and Efficient Off-Policy Reinforcement Learning

Retrace(\(\lambda\)) is a convergent off-policy multi-step algorithm extending the DQN agent.
Safe and Efficient Off-Policy Reinforcement Learning

The Retrace algorithm comes with the theoretical guarantee that in finite state and action spaces, repeatedly updating our current estimate Q produces a sequence of Q functions which converges to $Q^\pi$ for a fixed $\pi$ or to $Q^*$ if we consider a sequence of policies $\pi$ which become increasingly greedy w.r.t. the Q estimate.
Preliminary (Off-policy)

- Learning the state (action) value function for a policy $\pi$:
  \[ Q^\pi(x, a) = \mathbb{E}_{\pi} \left[ r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \mid x_0 = x, a_0 = a \right] \]

- You can learn optimal control if it is a greedy policy to the current estimate $Q(x; a)$ e.g. Q-learning
- **On-policy**: learning from data collected by $\pi$
- **Off-policy**: learning from data collected by $\mu \neq \pi$
- Off-policy methods have advantages:
  - Sample-efficient (e.g. experience replay)
  - Exploration by $\mu$

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Off-policy Learning

- Target policy ($\pi$): deterministic (optimal greedy)
- Behavior policy ($\mu$): stochastic (exploratory)
- Assumption of coverage: $\pi(a|s) > 0$ implies $\mu(a|s) > 0$

Preliminary (Importance Sampling)

Off-policy Learning

- **Target policy** ($\pi$) : deterministic (optimal greedy)
- **Behavior policy** ($\mu$) : stochastic (exploratory)
- **Assumption of coverage**: $\pi(a|s) > 0$ implies $\mu(a|s) > 0$
- **Importance sampling**:

\[
\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\]

Preliminary (Importance Sampling)

Off-policy Learning

- **Target policy** ($\pi$) : deterministic (optimal greedy)
- **Behavior policy** ($\mu$) : stochastic (exploratory)
- **Assumption of coverage**: $\pi(a|s) > 0$ implies $\mu(a|s) > 0$
- **Importance sampling**:

\[
\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\]

\[
\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)}
\]

Preliminary (Importance Sampling)

Off-policy Learning

- **Target policy** ($\pi$): deterministic (optimal greedy)
- **Behavior policy** ($\mu$): stochastic (exploratory)
- **Assumption of coverage**: $\pi(a|s) > 0$ implies $\mu(a|s) > 0$
- **Importance sampling**:

\[
\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\]

\[
\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)}
\]

Preliminary (Importance Sampling)

Off-policy Learning

- **Target policy** ($\pi$): deterministic (optimal greedy)
- **Behavior policy** ($\mu$): stochastic (exploratory)
- **Assumption of coverage**: $\pi(a|s) > 0$ implies $\mu(a|s) > 0$
- **Importance sampling**:

\[
\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1})
\]

\[
= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\]

\[
\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}.
\]

Preliminary (Importance Sampling)

Importance Sampling

- **Usage:**
  - **Wanted:** the expected returns (values) under the target policy: \( v_\pi(S_t) \)

Preliminary (Importance Sampling)

Importance Sampling

- **Usage:**
  - **Wanted:** the expected returns (values) under the target policy: $v_\pi(S_t)$
  - **Got:** Returns $G_t$ based on the wrong (behavior) policy:
    $$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$
    $$\mathbb{E}[G_t | S_t] = v_b(S_t)$$

Importance Sampling

- **Usage:**
  - **Wanted:** the expected returns (values) under the target policy: \( v_{\pi}(S_t) \)
  - **Got:** Returns \( G_t \) based on the wrong (behavior) policy:
    \[
    G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k
    \]
    \[
    \mathbb{E}[G_t | S_t] = v_b(S_t)
    \]
  - **Solution:** introduce the importance sampling (for discrepancy correction):
    \[
    \mathbb{E}[\rho_{t:T-1} G_t | S_t] = v_{\pi}(S_t)
    \]
    The ratio \( \rho(t:T-1) \) transforms the returns to have the right expected value.

Preliminary (Importance Sampling)

Importance Sampling

- **Usage:**
  - **Wanted:** the expected returns (values) under the target policy: \( v_\pi(S_t) \)
  - **Got:** Returns \( G_t \) based on the wrong (behavior) policy:
    \[
    G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k 
    \]
    \[
    \mathbb{E}[G_t | S_t] = v_b(S_t) 
    \]
  - **Solution:** introduce the importance sampling:
    \[
    \mathbb{E}[\rho_{t:T-1} G_t | S_t] = v_\pi(S_t) 
    \]
    The ratio \( \rho(t:T-1) \) transforms the returns to have the right expected value

Preliminary (Importance Sampling)

Importance Sampling

● Problem of variances:
  ○ Example: an episodes has 100 steps and $\gamma = 0$.
    The return from time 0 will then be just $G_0 = R_1$

Importance Sampling

- **Problem of variances:**
  - **Example:** an episode has 100 steps and $\gamma = 0$.
    
    The return from time 0 will then be just $G_0 = R_1$
  
  - Its importance sampling ratio will be a product of 100 factors:

    $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

    $\frac{\pi(A_0|S_0)}{\mu(A_0|S_0)} \frac{\pi(A_1|S_1)}{\mu(A_1|S_1)} \cdots \frac{\pi(A_{99}|S_{99})}{\mu(A_{99}|S_{99})}$

Preliminary (Importance Sampling)

Importance Sampling

- **Problem of variances:**
  - **Example:** an episode has 100 steps and $\gamma = 0$. The return from time 0 will then be just $G_0 = R_1$
  - Its importance sampling ratio will be a product of 100 factors:
    $$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$
    $$\frac{\pi(A_0|S_0)}{\mu(A_0|S_0)} \cdot \frac{\pi(A_1|S_1)}{\mu(A_1|S_1)} \cdots \frac{\pi(A_{99}|S_{99})}{\mu(A_{99}|S_{99})}$$
  - But it is really only necessary to scale by the first factor. The other 99 factors are irrelevant, but they add enormously to its variance.

Preliminary (N-steps Returns)

N-step TD Prediction

- Monte Carlo Return:
  \[ G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T, \]

- One Step Return:
  \[ G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1}), \]

- N steps Return:
  \[ G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \]

Preliminary (N-steps Returns)

N-step TD Prediction

- Monte Carlo Return:
  \[ G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T, \]

- One Step Return:
  \[ G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1}), \]

- N steps Return:
  \[ G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \]

- The natural state-value learning algorithm for using n-step returns
  \[ V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha [G_{t:t+n} - V_{t+n-1}(S_t)], \quad 0 \leq t < T, \]

Preliminary (λ-steps Returns)

The λ-return

An alternative way of moving smoothly between Monte Carlo and one-step TD methods

\[ G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}. \]

\[ G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t. \]

Preliminary (λ-steps Returns)

The λ-return

An alternative way of moving smoothly between Monte Carlo and one-step TD methods

\[
G_λ^T = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n},
\]

\[
G_λ^T = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t,
\]

The \( \lambda \)-return

- The \( \lambda \)-return could be written as:

\[
R^\lambda_t = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t
\]

Until termination \enspace After termination

- If \( \lambda = 1 \), you get MC return:

\[
R^\lambda_t = (1 - 1) \sum_{n=1}^{T-t-1} 1^{n-1} R_t^{(n)} + 1^{T-t-1} R_t = R_t
\]

- If \( \lambda = 0 \), you get TD(0):

\[
R^\lambda_t = (1 - 0) \sum_{n=1}^{T-t-1} 0^{n-1} R_t^{(n)} + 0^{T-t-1} R_t = R_t^{(1)}
\]

Ref: http://www-anw.cs.umass.edu/~barto/courses/cs687/Chapter%207.pdf
Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Preliminary (λ-steps Returns)

The λ-return

- The λ-return could be written as:
  \[
  R_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t
  \]
  Until termination + After termination

- If \( \lambda = 1 \), you get MC return:
  \[
  R_t^\lambda = (1 - 1) \sum_{n=1}^{T-t-1} 1^{n-1} R_t^{(n)} + 1^{T-t-1} R_t = R_t
  \]

- If \( \lambda = 0 \), you get TD(0):
  \[
  R_t^\lambda = (1 - 0) \sum_{n=1}^{T-t-1} 0^{n-1} R_t^{(n)} + 0^{T-t-1} R_t = R_t^{(1)}
  \]

Questions:

- Can we apply to Q learning?
  - Policy evaluation:
    estimate \( Q^\pi \) from samples collected by \( \mu \)
  - Control:
    estimate \( Q^* \) from samples collected by \( \mu \)

- Possible solution
  - Watkins’s Q(\( \lambda \)) [Watkins 1989] method
  - Cut off traces whenever a non-greedy action is taken
  - Converges to \( Q^* \) under a mild assumption (first proved in Retrace paper)

Ref: http://www-anw.cs.umass.edu/~barto/courses/cs687/Chapter%207.pdf
Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Preliminary (λ-steps Returns)

**Watkins’s Q(λ)**

Classic multi-step algorithm for off-policy control

This approach is an off-policy eligibility trace which updates more than one Q-value per step.

This can result in a significant increase in the speed of learning at a cost to stability

unproven of convergence until Retrace (2016, ~30 years)

Safe and Efficient Off-Policy Reinforcement Learning
Retrace

Safe and Efficient Off-Policy Reinforcement Learning

- Proposes a new off-policy multi-step RL method: $\text{Retrace}(\lambda)$

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Retrace

Safe and Efficient Off-Policy Reinforcement Learning

- Proposes a new off-policy multi-step RL method: **Retrace**(λ)
- Theoretical advantages
  - It converges for any \( \pi, \mu \) (safe)
  - It makes the best use of samples if \( \pi \) and \( \mu \) are close to each other (efficient)
  - Its variance is lower than importance sampling
Retrace

Safe and Efficient Off-Policy Reinforcement Learning

- Proposes a new off-policy multi-step RL method: \textbf{Retrace}(\lambda)
- Theoretical advantages
  - It converges for any \(\pi, \mu\) (safe)
  - It makes the best use of samples if \(\pi\) and \(\mu\) are close to each other (efficient)
  - Its variance is lower than importance sampling
- Empirical evaluation
  - On Atari 2600, it beats one-step Q-learning (DQN) and the existing multi-step methods (Q*(\lambda), Tree-Backup)

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Retrace

Safe and Efficient Off-Policy Reinforcement Learning

- Proposes a new off-policy multi-step RL method: Retrace(\(\lambda\))
- Theoretical advantages
  - It converges for any \(\pi, \mu\) (safe)
  - It makes the best use of samples if \(\pi\) and \(\mu\) are close to each other (efficient)
  - Its variance is lower than importance sampling
- Empirical evaluation
  - On Atari 2600, it beats one-step Q-learning (DQN) and the existing multi-step methods (Q*(\(\lambda\)), Tree-Backup)
- Proves the convergence of Watkins's Q(\(\lambda\)) for the first time

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
On-policy multi-step methods

**TD(λ)**

Behavior policy $\mu(a|x)$

- A popular multi-step algorithm for on-policy policy evaluation
- $\Delta_t Q(x, a) = (\gamma \lambda)^t \delta_t$ where $\lambda \in [0,1]$ is chosen to balance bias and variance
- Multi-step methods have advantages:
  - Rewards are propagated **rapidly**
  - Bias introduced by bootstrapping is **reduced**

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Off-policy multi-step methods

Behavior policy $\mu(a|x)$
Target policy $\pi(a|x)$

$\delta_t = r_t + \gamma \mathbb{E}_\pi Q(x_{t+1}, \cdot) - Q(x_t, a_t)$

Can you use $\delta_t$ to estimate $Q^\pi(x_t, a_t)$ for all $s \leq t$?

- Three methods mentioned in the paper:

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Off-policy multi-step methods

Behavior policy $\mu(a|x)$
Target policy $\pi(a|x)$

$\delta_t = r_t + \gamma \mathbb{E}_{\pi} \left[ Q(x_{t+1}, \cdot) \right] - Q(x_t, a_t)$

Can you use $\delta_t$ to estimate $Q^\pi(x_t, a_t)$ for all $s \leq t$?

- Three methods mentioned in the paper:
  - Importance Sampling (IS) [Precup et al. 2000]
  - $Q^\pi(\lambda)$ [Harutyunyan et al. 2016]
  - Tree-Backup (TB) [Precup et al. 2000]

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Off-policy multi-step methods: Importance Sampling (IS) [Precup et al. 2000]

Pros: Unbiased estimate of $Q^\pi$

Cons: Large variance since $\frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$ is not bounded

From the presentation by the authors: https://ewrl.files.wordpress.com/2016/12/munos.pdf

\[ \Delta_t Q(x, a) = \gamma^t (\prod_{1 \leq s \leq t} \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}) \delta_t \]
Off-policy multi-step methods: Importance Sampling (IS) [Precup et al. 2000]

Pros: Unbiased estimate of $Q^\pi$

Cons: Large variance since $\frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$ is not bounded (not efficient)

Reweight the trace by the product of IS ratios

$$\Delta_t Q(x, a) = \gamma^t \left( \prod_{1 \leq s \leq t} \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \right) \delta_t$$
Off-policy multi-step methods: $Q^\pi(\lambda)$ [Harutyunyan et al. 2016]

- **Pros:**
  - Convergent if $\pi$ and $\mu$ are sufficiently close to each other or $\lambda$ is sufficiently small: $\lambda < \frac{1-\gamma}{\gamma \epsilon}$, where $\epsilon := \max_x \| \pi(\cdot|x) - \mu(\cdot|x) \|_1$

- **Cons:**
  - Not convergent otherwise

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Off-policy multi-step methods: $Q^{\pi}(\lambda)$ [Harutyunyan et al. 2016]

Pros:
- Convergent if $\pi$ and $\mu$ are sufficiently close to each other or $\lambda$ is sufficiently small: $\lambda < \frac{1-\gamma}{\gamma \epsilon}$, where $\epsilon := \max_x \|\pi(\cdot|x) - \mu(\cdot|x)\|_1$

Cons:
- Not convergent otherwise (not safe)

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Off-policy multi-step methods: Tree-Backup (TB) [Precup et al. 2000]

Pros:
- Convergent for any $\pi$ and $\mu$. even if $\mu$ is unknown and/or non-Markov

Cons:
- $\prod_{1 \leq s \leq t} \pi(a_s|x_s)$ decays rapidly when near on-policy

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Off-policy multi-step methods: Tree-Backup (TB) [Precup et al. 2000]

Behavior policy $\mu(a|x)$
Target policy $\pi(a|x)$

Pros:

- Convergent for any $\pi$ and $\mu$, even if $\mu$ is unknown and/or non-Markov

Cons:

- Decays rapidly when near on-policy (not efficient)

From the presentation by the authors: https://ewrl.files.wordpress.com/2016/12/munos.pdf

$\Delta_t Q(x, a) = (\gamma \lambda)^t (\prod_{1 \leq s \leq t} \pi(a_s|x_s)) \delta_t$

Reweight the traces by the product of target probabilities

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Retrace

General off-policy return-based algorithm

\[ \Delta Q(x, a) = \sum_{t \geq 0} \gamma^t \left( \prod_{1 \leq s \leq t} c_s \right) \left( r_t + \gamma \mathbb{E}_\pi Q(x_{t+1}, \cdot) - Q(x_t, a_t) \right)_{\delta_t} \]

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Retrace

General off-policy return-based algorithm

\[ \Delta Q(x, a) = \sum_{t \geq 0} \gamma^t \left( \prod_{1 \leq s \leq t} c_s \right) \left( r_t + \gamma \mathbb{E}_\pi Q(x_{t+1}, \cdot) - Q(x_t, a_t) \right) \]

<table>
<thead>
<tr>
<th>Importance sampling</th>
<th>Definition of $c_s$</th>
<th>Estimation variance</th>
<th>Guaranteed convergence $\dagger$</th>
<th>Use full returns (near on-policy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importance sampling</td>
<td>$\frac{\pi(a_s</td>
<td>x_s)}{\mu(a_s</td>
<td>x_s)}$</td>
<td>High</td>
</tr>
<tr>
<td>$Q^\pi(\lambda)$</td>
<td>$\lambda$</td>
<td>Low</td>
<td>for $\pi$ close to $\mu$</td>
<td>yes</td>
</tr>
<tr>
<td>TB($\lambda$)</td>
<td>$\lambda \pi(a_s</td>
<td>x_s)$</td>
<td>Low</td>
<td>for any $\pi, \mu$</td>
</tr>
</tbody>
</table>

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
General off-policy return-based algorithm

\[
\Delta Q(x, a) = \sum_{t \geq 0} \gamma^t \left( \prod_{1 \leq s \leq t} c_s \right) \left( r_t + \gamma \mathbb{E}_{\pi} Q(x_{t+1}, \cdot) - Q(x_t, a_t) \right)
\]

|                          | Definition of \( c_s \) | Estimation variance | Guaranteed convergence\(|\) | Use full returns (near on-policy) |
|--------------------------|--------------------------|----------------------|-----------------------------|---------------------------------|
| Importance sampling      | \( \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \) | High                 | for any \( \pi, \mu \)     | yes                             |
| \( Q^\pi(\lambda) \)    | \( \lambda \)            | Low                  | for \( \pi \) close to \( \mu \) | yes                             |
| TB(\(\lambda\))         | \( \lambda \pi(a_s|x_s) \) | Low                  | for any \( \pi, \mu \)      | no                              |

- None of the existing methods is perfect (low variance, safe and efficient)
  - Safe: i.e. convergent for any \( \pi \) and \( \mu \) (Q(\(\lambda\)))
  - Efficient: i.e. using full returns when on-policy (Tree-Backup)

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Retrace

Proposed Solution: Retrace(\lambda)

Retrace

Proposed Solution: Retrace(\(\lambda\))

\[ c_s = \lambda \min \left( 1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \right) \]
Retrace

Proposed Solution: Retrace(\(\lambda\))

\[
c_s = \lambda \min \left(1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \right)
\]

Properties:

- Low variance: since \(c_s \leq 1\)
- Safe (off policy): cut the traces when needed \(c_s \in \left[0, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \right]\)
- Efficient (on policy): keep the traces near on policy. Note that \(c_s \geq \lambda \pi(a_s|x_s)\)

## Retrace

**Proposed Solution: Retrace(\(\lambda\))**

<table>
<thead>
<tr>
<th>Method</th>
<th>Definition of (c_s)</th>
<th>Estimation variance</th>
<th>Guaranteed convergence†</th>
<th>Use full returns (near on-policy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importance sampling</td>
<td>(\frac{\pi(a_s</td>
<td>x_s)}{\mu(a_s</td>
<td>x_s)})</td>
<td>High</td>
</tr>
<tr>
<td>(Q^\pi(\lambda))</td>
<td>(\lambda)</td>
<td>Low</td>
<td>for (\pi) close to (\mu)</td>
<td>yes</td>
</tr>
<tr>
<td>TB((\lambda))</td>
<td>(\lambda \pi(a_s</td>
<td>x_s))</td>
<td>Low</td>
<td>for any (\pi, \mu)</td>
</tr>
<tr>
<td>Retrace((\lambda))</td>
<td>(\lambda \min (1, \frac{\pi(a_s</td>
<td>x_s)}{\mu(a_s</td>
<td>x_s)}))</td>
<td>Low</td>
</tr>
</tbody>
</table>

Retrace

**Off-policy policy evaluation**

**Theorem-1**: Assume finite state space. Generate trajectories according to behavior policy $\mu$. Update all states along trajectories according to

$$\Delta Q(x, a) = \sum_{t \geq 0} \gamma^t \left( \prod_{1 \leq s \leq t} c_s \right) \left( r_t + \gamma \mathbb{E}_\pi Q(x_{t+1}, \cdot) - Q(x_t, a_t) \right)$$

Off-policy policy evaluation

**Theorem-1:** Assume finite state space. Generate trajectories according to behavior policy \( \mu \). Update all states along trajectories according to

\[
Q_{k+1}(x, a) = Q_k(x, a) + \alpha_k \sum_{t \geq 0} \gamma^t (c_1 \ldots c_t) (r_t + \gamma \mathbb{E}_\pi Q_k(x_{t+1}, \cdot) - Q_k(x_t, a_t))
\]

Assume all states visited infinitely often.

Off-policy policy evaluation

**Theorem-1**: Assume finite state space. Generate trajectories according to behavior policy $\mu$. Update all states along trajectories according to

$$Q_{k+1}(x, a) = Q_k(x, a) + \alpha_k \sum_{t \geq 0} \gamma^t (c_1 \ldots c_t) (r_t + \gamma \mathbb{E}_\pi Q_k(x_{t+1}, \cdot) - Q_k(x_t, a_t))$$

Assume all states visited infinitely often.

If $0 \leq c_s \leq \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$ then $Q_k \to Q^\pi$

Sufficient conditions for a safe algorithm (works for any $\pi$ and $\mu$)
Retrace

Tradeoff for trace coefficients $C_s$

- Contraction coefficient of the expected operator

$$\eta := \gamma - (1 - \gamma) \mathbb{E}_\mu \left[ \sum_{t \geq 1} \gamma^t (c_1 \cdots c_t) \right] \in [0, \gamma]$$

  - $\eta = \gamma$ when $c_s = 0$ \hspace{0.5cm} \text{(one-step Bellman update)}
  - $\eta = 0$ when $c_s = 1$ \hspace{0.5cm} \text{(full Monte-Carlo rollouts)}

- Variance of the estimate (can be infinite for $c_s = $ case)

  - Large $c_s$: uses multi-steps returns, but large variance
  - Small $c_s$: low variance, but do not use multi-steps returns

Retrace

Retrace(\(\lambda\)) for optimal control

Let \((\mu_k)\) and \((\pi_k)\) sequences of behavior and target policies and:

\[
Q_{k+1}(x, a) = Q_k(x, a) + \alpha_k \sum_{t \geq 0} (\lambda \gamma)^t \prod_{1 \leq s \leq t} \min \left( 1, \frac{\pi_k(a_s|s)}{\mu_k(a_s|s)} \right) (r_t + \gamma \mathbb{E}_{\pi} Q_k(x_{t+1}, \cdot) - Q_k(x_t, a_t))
\]

Retrace

Retrace(λ) for optimal control
Let \((\mu_k)\) and \((\pi_k)\) sequences of behavior and target policies and:

\[
Q_{k+1}(x, a) = Q_k(x, a) + \alpha_k \sum_{t \geq 0} (\lambda \gamma)^t \prod_{1 \leq s \leq t} \min \left( 1, \frac{\pi_k(a_s|x_s)}{\mu_k(a_s|x_s)} \right) \left( r_t + \gamma \mathbb{E}_\pi Q_k(x_{t+1}, \cdot) - Q_k(x_t, a_t) \right)
\]

Theorem 2

Under previous assumptions

Assume \((\pi_k)\) are “increasingly greedy” wrt \((Q_k)\)

Then, a.s., \(Q_k \rightarrow Q^*\)

Remarks

- If \((\pi_k)\) are greedy policies, then \(c_s = \lambda \mathbb{1}\{a_s \in \arg\max_a Q_k(x_s, a)\}\)

  \[\rightarrow \text{Convergence of Watkin's } Q(\Lambda) \text{ to } Q^*\]

  (open problem since 1989)

Under assumption of finite-state space:

- Convergence to optimal policy
- Cut traces when -and only when- needed
- Adjust the length of the backup to the “off-policy-ness” of the data

Retrace for deep RL

Several actor-critic architectures at DeepMind:


- **Reactor** (Retrace-actor) [Gruesly et al., 2018]. Use beta-LOO to update policy. Use LSTM.

- **MPO** (Maximum a posteriori Policy Optimization) [Abdolmaleki et al., 2018] Soft (KL-regularized) policy improvement.

- **IMPALA** (IMPortance Weighted Actor-Learner Architecture) [Espeholt et al., 2018]. Heavily distributed agent. Uses V-trace.

Retrace

Evaluation on Atari 2600

- Performance comparison:
  - Inter-algorithm scores are normalized so that 0 and 1 respectively correspond to the worst and best scores for a particular game (Roughly, a strictly higher curve corresponds to a better algorithm)
  - Retrace(λ) performs best on 30 out of 60 games

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Evaluation on Atari 2600: Retrace vs DQN

Games: (Blue: DQN  Red: Retrace)
Asteroids, Defender, Demon Attack, Hero, Krull, River Raid, Space Invaders, Star Gunner, Wizard of Wor, Zaxxon

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Retrace Evaluation on Atari 2600

- Sensitivity to the value of $\lambda$:
  - Retrace($\lambda$) is robust and consistently outperforms Tree-Backup
  - $Q^*$ performs best for small values of $\lambda$
  - Note that the Q-learning scores are fixed across different $\lambda$

Ref: slides from Yasuhiro Fujita, Preferred Networks Inc.
Conclusions

- General update rule for off-policy return-based RL
- Conditions under which an algo is safe and efficient
- We recommend to use **Retrace**:
  - Converges to Q* (finite state/action space, policy $\pi$ is increasingly greedy)
  - Safe: cut the traces when needed
  - Efficient: but only when needed
  - Works for policy evaluation and for control
  - Particularly suited for deep RL
- Extensions:
  - Works in continuous action spaces
  - Can be used in off-policy policy-gradient [Wang et al., 2016]
A fast and sample-efficient Actor-Critic agent for Reinforcement Learning (Reactor)
A fast and sample-efficient Actor-Critic agent for Reinforcement Learning (Reactor)

[Contributions]

- **Sample-efficiency:**
  Higher than Prioritized Dueling DQN (Wang et al., 2017) and Categorical DQN (Bellemare et al., 2017)

- **Time-efficiency:**
  Better run-time performance than A3C (Mnih et al., 2016).
A fast and sample-efficient Actor-Critic agent for Reinforcement Learning (Reactor)

[Contributions]

- **Sample-efficiency:**
  Higher than Prioritized Dueling DQN (Wang et al., 2017) and Categorical DQN (Bellemare et al., 2017)

- **Time-efficiency:**
  Better run-time performance than A3C (Mnih et al., 2016).

[Reactor (Retrace-Actor)]
Combining the sample-efficiency of off-policy experience replay with the time-efficiency of asynchronous algorithms
The Reactor is a combination of four novel contributions on top of recent improvements to both deep value-based RL and policy-gradient algorithms.

- **β-leave-one-out:**
  Improves the trade-off between variance and bias by using action values as a baseline.

- **Distributional Retrace:**
  Brings multi-step off-policy updates to the distributional reinforcement learning setting.

- **Prioritized sequences replay:**
  Present the lazy initialization for more efficient replay prioritization.

- **Agent Architecture:**
  Propose an optimized network and parallel training architecture.
Reactor

β-leave-one-out

- Need a policy gradient algorithm to train the actor policy $\pi$ based on current estimate $Q(x, a)$ of $Q^\pi(x, a)$:

$$\nabla V^\pi(x_0) = \mathbb{E} \left[ \sum_t \gamma^t \sum_a Q^\pi(x_t, a) \nabla \pi(a|x_t) \right].$$
β-leave-one-out

- Need a policy gradient algorithm to train the actor policy $\pi$ based on current estimate $Q(x, a)$ of $Q^\pi(x, a)$:

$$\nabla V^\pi(x_0) = \mathbb{E}\left[ \sum_t \gamma^t \sum_a Q^\pi(x_t, a) \nabla \pi(a|x_t) \right].$$

- Simplify the notations (find a way to estimate gradient $G$):

$$G = \sum_a Q^\pi(a) \nabla \pi(a).$$
Reactor

β-leave-one-out

- Need a policy gradient algorithm to train the actor policy $\pi$ based on current estimate $Q(x, a)$ of $Q^\pi(x, a)$:

$$
\nabla V^\pi(x_0) = \mathbb{E}\left[ \sum_t \gamma^t \sum_a Q^\pi(x_t, a)\nabla \pi(a|x_t) \right].
$$

- Simplify the notations (find a way to estimate gradient $G$):

$$
G = \sum_a Q^\pi(a)\nabla \pi(a).
$$

- Unbiased estimate of $G$ (sampled from behaviour policy $\mu$ with IS ratio):

$$
\hat{G}_{ISLR} = \frac{\pi(\hat{a})}{\mu(\hat{a})}(R(\hat{a}) - V)\nabla \log \pi(\hat{a})
$$
Reactor

β-leave-one-out

• Need a policy gradient algorithm to train the actor policy $\pi$ based on current estimate $Q(x, a)$ of $Q^\pi(x, a)$:

$$\nabla V^\pi(x_0) = \mathbb{E}\left[\sum_t \gamma^t \sum_a Q^\pi(x_t, a) \nabla \pi(a|x_t)\right].$$

• Simplify the notations (find a way to estimate gradient $G$):

$$G = \sum_a Q^\pi(a) \nabla \pi(a).$$

• Unbiased estimate of $G$ (sampled from behaviour policy $\mu$ with IS ratio):

$$\hat{G}_{ISLR} = \frac{\pi(\hat{a})}{\mu(\hat{a})} (R(\hat{a}) - V(\hat{a}) \nabla \log \pi(\hat{a}))$$

Baseline depends on the state
Reactor

β-leave-one-out

- Need a policy gradient algorithm to train the actor policy $\pi$ based on current estimate $Q(x, a)$ of $Q^\pi(x, a)$:
  \begin{equation}
  \nabla V^\pi(x_0) = \mathbb{E}\left[ \sum_t \gamma^t \sum_a Q^\pi(x_t, a) \nabla \pi(a|x_t) \right].
  \end{equation}

- Simplify the notations (find a way to estimate gradient $G$):
  \begin{equation}
  G = \sum_a Q^\pi(a) \nabla \pi(a).
  \end{equation}

- Unbiased estimate of $G$ (sampled from behaviour policy $\mu$ with IS ratio):
  \begin{equation}
  \hat{G}_{\text{ISLR}} = \frac{\pi(\hat{a})}{\mu(\hat{a})} (R(\hat{a}) - V) \nabla \log \pi(\hat{a})
  \end{equation}

  Unbiased, but high variance, needs reducing!
\[ \hat{G}_{\text{LOO}} = R(\hat{a}) \nabla \pi(\hat{a}) + \sum_{a \neq \hat{a}} Q(a) \nabla \pi(a). \]
Reactor

\(\beta\)-leave-one-out

- leave-one-out (LOO) estimate of \(G\):
  Instead of applying IS, estimate \(G\) directly from the return \(R(a)\) for the chosen action \(a\) and current estimate \(Q(x, a)\) of \(Q^\pi(x, a)\)

\[
\hat{G}_{LOO} = R(\hat{a}) \nabla \pi(\hat{a}) + \sum_{a \neq \hat{a}} Q(a) \nabla \pi(a).
\]

Low variance
β-leave-one-out

- leave-one-out (LOO) estimate of $G$:
  Instead of applying IS, estimate $G$ directly from the return $R(a)$ for the chosen action $a$ and current estimate $Q(x, a)$ of $Q^\pi(x, a)$

$$
\hat{G}_{LOO} = R(\hat{a}) \nabla \pi(\hat{a}) + \sum_{a \neq \hat{a}} Q(a) \nabla \pi(a).
$$

but may be biased if the estimated $Q(x, a)$ values differ from $Q^\pi(x, a)$
Reactor

**β-leave-one-out**

- leave-one-out (LOO) estimate of $G$:
  Instead of applying IS, estimate $G$ directly from the return $R(a)$ for the chosen action $a$ and current estimate $Q(x, a)$ of $Q^\pi(x, a)$

$$
\hat{G}_{\text{LOO}} = R(\hat{a}) \nabla \pi(\hat{a}) + \sum_{a \neq \hat{a}} Q(a) \nabla \pi(a).
$$

- A better bias-variance tradeoff --> $\beta$-LOO policy-gradient estimate:
Reactor

β-leave-one-out

- leave-one-out (LOO) estimate of G:
  Instead of applying IS, estimate G directly from the return R(a) for the chosen action a and current estimate $Q(x, a)$ of $Q^\pi(x, a)$

  \[ \hat{G}_{\text{LOO}} = R(\hat{a}) \nabla \pi(\hat{a}) + \sum_{a \neq \hat{a}} Q(a) \nabla \pi(a). \]

- A better bias-variance tradeoff --> $\beta$-LOO policy-gradient estimate:

  \[ \hat{G}_{\beta-\text{LOO}} = \beta (R(\hat{a}) - Q(\hat{a})) \nabla \pi(\hat{a}) + \sum_a Q(a) \nabla \pi(a), \]
Reactor

β-leave-one-out

- leave-one-out (LOO) estimate of $G$:
  Instead of applying IS, estimate $G$ directly from the return $R(a)$ for the chosen action $a$ and current estimate $Q(x, a)$ of $Q^\pi(x, a)$

$$\hat{G}_{\text{LOO}} = R(\hat{a}) \nabla \pi(\hat{a}) + \sum_{a \neq \hat{a}} Q(a) \nabla \pi(a).$$

- A better bias-variance tradeoff --> β-LOO policy-gradient estimate:

$$\hat{G}_{\beta-\text{LOO}} = \beta (R(\hat{a}) - Q(\hat{a})) \nabla \pi(\hat{a}) + \sum_a Q(a) \nabla \pi(a),$$

where $\beta = \beta(\mu, \pi, a)$ can be a function of both policies, $\pi$ and $\mu$, and the selected action $a$
Reactor

β-leave-one-out

- Property of β-LOO for given β:

General case: \( \hat{G}_{β-LOO} = β(R(\hat{a}) - Q(\hat{a}))\nabla \pi(\hat{a}) + \sum_a Q(a)\nabla \pi(a), \)
Reactor

β-leave-one-out

- Property of β-LOO for given β:

\[
\hat{G}_{\text{LOO}} = R(\hat{a}) \nabla \pi(\hat{a}) + \sum_{a \neq \hat{a}} Q(a) \nabla \pi(a).
\]
Reactor

β-leave-one-out

- Property of β-LOO for given β:

When $\beta = 1/\mu$:

$$\hat{\mathcal{G}}_{\frac{1}{\mu}}^{\text{LOO}} = \frac{\pi(\hat{a})}{\mu(\hat{a})} (R(\hat{a}) - Q(\hat{a})) \nabla \log \pi(\hat{a}) + \sum_a Q(a) \nabla \pi(a).$$
Reactor

β-leave-one-out

- Property of β-LOO for given β:
  When $\beta = 1/\mu$:
  $$\hat{G}_{1/\mu, \text{LOO}} = \frac{\pi(\hat{a})}{\mu(\hat{a})} (R(\hat{a}) - Q(\hat{a})) \nabla \log \pi(\hat{a}) + \sum_a Q(a) \nabla \pi(a).$$

- Choice of β:
  - Low bias: as $\beta(a)$ is close to $1/\mu(a)$ or $Q(x, a)$ close to $Q^\pi(x, a)$.
  - Unbiased: as $\beta(a)$ is equal to $1/\mu(a)$
  - Low Variance: as $\beta(a)$ is small

- Bias-Variance tradeoff:
  - Choose $\beta(\hat{a}) = \min \left( c, \frac{1}{\mu(\hat{a})} \right)$ for some constant $c \geq 1$
Reactor

Distributional Retrace
Extend C51 to multi-step Bellman backup.

- The n-step distributional Bellman target:

\[
\sum_i q_i(x_{t+n}, a) \delta_{z_i}^n, \text{ with } z_i^n = \sum_{s=t}^{t+n-1} \gamma^{s-t} r_s + \gamma^n z_i
\]

- The expectation is:

\[
\sum_{s=t}^{t+n-1} \gamma^{s-t} r_s + \gamma^n Q(x_{t+n}, a)
\]
Reactor

Distributional Retrace

- Original Retrace:

\[ \Delta Q(x_t, a_t) \overset{\text{def}}{=} \sum_{s \geq t} \gamma^{s-t} (c_{t+1} \cdots c_s) \delta^\pi_s Q \]
Reactor

Distributional Retrace

- Original Retrace:

\[ \Delta Q(x_t, a_t) \overset{\text{def}}{=} \sum_{s \geq t} \gamma^{s-t} (c_{t+1} \ldots c_s) \delta_s^\pi Q \]

- Distributional Retrace:

\[ \Delta Q(x_t, a_t) = \sum_{n \geq 1} \sum_{a \in A} \alpha_{n,a} \left[ \sum_{s=t}^{t+n-1} \gamma^{s-t} r_s + \gamma^n Q(x_{t+n}, a) \right] - Q(x_t, a_t) \]

where \( \alpha_{n,a} = (c_{t+1} \ldots c_{t+n-1}) (\pi(a|x_{t+n}) - \mathbb{I}\{a = a_{t+n}\}c_{t+n}) \)
Reactor

Distributional Retrace

- A mixture of n-step distribution (Retrace target distribution):

\[
\sum_{i=1}^{\infty} q_i^* (x_t, a_t) \delta_{z_i}, \text{ with } q_i^* (x_t, a_t) = \sum_{n \geq 1} \sum_{a} \alpha_{n,a} \sum_{j} q_j (x_{t+n}, a_{t+n}) h_{z_i} (z_{j}^n)
\]
Reactor

Distributional Retrace

- A mixture of $n$-step distribution (Retrace target distribution):
  \[
  \sum_{i=1}^{\infty} q_i^*(x_t, a_t) \delta_{z_i}, \quad \text{with} \quad q_i^*(x_t, a_t) = \sum_{n \geq 1} \sum_{a} \alpha_{n,a} \sum_{j} q_j(x_{t+n}, a_{t+n}) h_{z_i}(z_j^n)
  \]

- Update the current probabilities by performing a gradient step on the KL-Loss:
  \[
  \nabla \text{KL}(q^*(x_t, a_t), q(x_t, a_t)) = -\sum_{i=1}^{\infty} q_i^*(x_t, a_t) \nabla \log q_i(x_t, a_t)
  \]
Reactor

Distributional Retrace

- A mixture of n-step distribution (Retrace target distribution):
  \[ \sum_{i=1}^{n} q_i^*(x_t, a_t) \delta_{x_i}, \text{ with } q_i^*(x_t, a_t) = \sum_{n \geq 1} \sum_{a} \alpha_{n,a} \sum_{j} q_j(x_{t+n}, a_{t+n}) h_{z_i}(z_j^n) \]

- Update the current probabilities by performing a gradient step on the KL-Loss:
  \[ \nabla \text{KL}(q_i^*(x_t, a_t), q(x_t, a_t)) = - \sum_{i=1} q_i^*(x_t, a_t) \nabla \log q_i(x_t, a_t) \]

- Distributional Retrace is a **linear combination** of n-step Bellman backups
Reactor

Distributional Retrace

1. Mix action-value distributions by $\pi$

2. Shrink mixed distribution by $\gamma$

3. Shift distribution by $r_t$

4. Obtain target probabilities

Single Step (C51)
Reactor

Distributional Retrace

Multi Steps
Distributional Retrace
Prioritized sequences replay

- Prioritized experience replay adds new transitions to the replay buffer with a constant priority.

- Propose a way to add experience to the buffer with no priority, inserting a priority only after the transition has been sampled and used for training.

- Also, instead of sampling transitions, we assign priorities to all (overlapping) sequences of length $n$.

- When sampling, sequences with an assigned priority are sampled proportionally to that priority.
Reactor

Architecture

DQN

A3C

Reactor

Agent architecture

Decouple agent training:
Action-learning pair

Network architecture
Reactor

Architecture

DQN

A3C

Reactor

Agent architecture

Thread for action or learning

Network architecture
Reactor

Architecture

**DQN**

- act
- act
- act
- act
- learn
- act

**A3C**

- act
- act
- act
- act
- learn
- act
- act
- act
- act
- act
- learn
- large batch

**Agent architecture**

**Worker for action-learning pair**

**Network architecture**
Instead of stacking frame, using RNN
Reactor

Architecture

DQN

A3C

Agent architecture

Network architecture

Gradient block

For stability
TISLR -> add β-LOO -> add Prioritization -> add distributional
Reactor

Experiments

Reactor (10+1) means:
- 10 workers for action-learner pair
- 1 worker for shared parameter server (for network)
# Reactor

## Experiments

### Reactor performances on Atari

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Normalized Scores</th>
<th>Mean Rank</th>
<th>ELO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RANDOM</strong></td>
<td>0.00</td>
<td>11.65</td>
<td>-563</td>
</tr>
<tr>
<td><strong>HUMAN</strong></td>
<td>1.00</td>
<td>6.82</td>
<td>0</td>
</tr>
<tr>
<td><strong>DQN</strong></td>
<td>0.69</td>
<td>9.05</td>
<td>-172</td>
</tr>
<tr>
<td><strong>DDQN</strong></td>
<td>1.11</td>
<td>7.63</td>
<td>-58</td>
</tr>
<tr>
<td><strong>DUEL</strong></td>
<td>1.17</td>
<td>6.35</td>
<td>32</td>
</tr>
<tr>
<td><strong>PRIOR</strong></td>
<td>1.13</td>
<td>6.63</td>
<td>13</td>
</tr>
<tr>
<td><strong>PRIOR. DUEL.</strong></td>
<td>1.15</td>
<td>6.25</td>
<td>40</td>
</tr>
<tr>
<td><strong>A3C LSTM</strong></td>
<td>1.13</td>
<td>6.30</td>
<td>37</td>
</tr>
<tr>
<td><strong>RAINBOW</strong></td>
<td>1.53</td>
<td>4.18</td>
<td>186</td>
</tr>
<tr>
<td><strong>REACTOR ND</strong></td>
<td>1.51</td>
<td>4.98</td>
<td>126</td>
</tr>
<tr>
<td><strong>REACTOR</strong></td>
<td>1.65</td>
<td>4.58</td>
<td>156</td>
</tr>
<tr>
<td><strong>REACTOR 500M</strong></td>
<td><strong>1.82</strong></td>
<td><strong>3.65</strong></td>
<td><strong>227</strong></td>
</tr>
</tbody>
</table>

**Table 1:** Random human starts

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Normalized Scores</th>
<th>Mean Rank</th>
<th>ELO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RANDOM</strong></td>
<td>0.00</td>
<td>10.93</td>
<td>-673</td>
</tr>
<tr>
<td><strong>HUMAN</strong></td>
<td>1.00</td>
<td>6.89</td>
<td>0</td>
</tr>
<tr>
<td><strong>DQN</strong></td>
<td>0.79</td>
<td>8.65</td>
<td>-167</td>
</tr>
<tr>
<td><strong>DDQN</strong></td>
<td>1.18</td>
<td>7.28</td>
<td>-27</td>
</tr>
<tr>
<td><strong>DUEL</strong></td>
<td>1.51</td>
<td>5.19</td>
<td>143</td>
</tr>
<tr>
<td><strong>PRIOR</strong></td>
<td>1.24</td>
<td>6.11</td>
<td>70</td>
</tr>
<tr>
<td><strong>PRIOR. DUEL.</strong></td>
<td>1.72</td>
<td>5.44</td>
<td>126</td>
</tr>
<tr>
<td><strong>ACER^6 500M</strong></td>
<td>1.9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>RAINBOW</strong></td>
<td><strong>2.31</strong></td>
<td>3.63</td>
<td>270</td>
</tr>
<tr>
<td><strong>REACTOR ND</strong></td>
<td>1.80</td>
<td>4.53</td>
<td>195</td>
</tr>
<tr>
<td><strong>REACTOR</strong></td>
<td>1.87</td>
<td>4.46</td>
<td>196</td>
</tr>
<tr>
<td><strong>REACTOR 500M</strong></td>
<td>2.30</td>
<td><strong>3.47</strong></td>
<td><strong>280</strong></td>
</tr>
</tbody>
</table>

**Table 2:** 30 random no-op starts.
Reactor

Experiments

Reactor performances on Atari

Rainbow in no-op case is more sample efficiency, But may be overfitting

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Normalized Scores</th>
<th>Mean Rank</th>
<th>ELO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.00</td>
<td>11.65</td>
<td>-563</td>
</tr>
<tr>
<td>Human</td>
<td>1.00</td>
<td>6.82</td>
<td>0</td>
</tr>
<tr>
<td>DQN</td>
<td>0.69</td>
<td>9.05</td>
<td>-172</td>
</tr>
<tr>
<td>DDQN</td>
<td>1.11</td>
<td>7.63</td>
<td>-58</td>
</tr>
<tr>
<td>Duel</td>
<td>1.17</td>
<td>6.35</td>
<td>32</td>
</tr>
<tr>
<td>Prior</td>
<td>1.13</td>
<td>6.63</td>
<td>13</td>
</tr>
<tr>
<td>Prior. Duel.</td>
<td>1.15</td>
<td>6.25</td>
<td>40</td>
</tr>
<tr>
<td>A3C LSTM</td>
<td>1.13</td>
<td>6.30</td>
<td>37</td>
</tr>
<tr>
<td>Rainbow</td>
<td>1.53</td>
<td>4.18</td>
<td>186</td>
</tr>
<tr>
<td>Reactor ND 5</td>
<td>1.51</td>
<td>4.98</td>
<td>126</td>
</tr>
<tr>
<td>Reactor</td>
<td>1.65</td>
<td>4.58</td>
<td>156</td>
</tr>
<tr>
<td>Reactor 500M</td>
<td><strong>1.82</strong></td>
<td><strong>3.65</strong></td>
<td><strong>227</strong></td>
</tr>
</tbody>
</table>

**Table 1: Random human starts**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Normalized Scores</th>
<th>Mean Rank</th>
<th>ELO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.00</td>
<td>10.93</td>
<td>-673</td>
</tr>
<tr>
<td>Human</td>
<td>1.00</td>
<td>6.89</td>
<td>0</td>
</tr>
<tr>
<td>DQN</td>
<td>0.79</td>
<td>8.65</td>
<td>-167</td>
</tr>
<tr>
<td>DDQN</td>
<td>1.18</td>
<td>7.28</td>
<td>-27</td>
</tr>
<tr>
<td>Duel</td>
<td>1.51</td>
<td>5.19</td>
<td>143</td>
</tr>
<tr>
<td>Prior</td>
<td>1.24</td>
<td>6.11</td>
<td>70</td>
</tr>
<tr>
<td>Prior. Duel.</td>
<td>1.72</td>
<td>5.44</td>
<td>126</td>
</tr>
<tr>
<td>A3C LSTM</td>
<td>1.9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Rainbow</td>
<td><strong>2.31</strong></td>
<td>3.63</td>
<td>270</td>
</tr>
<tr>
<td>Reactor ND 5</td>
<td>1.80</td>
<td>4.53</td>
<td>195</td>
</tr>
<tr>
<td>Reactor</td>
<td>1.87</td>
<td>4.46</td>
<td>196</td>
</tr>
<tr>
<td>Reactor 500M</td>
<td>2.30</td>
<td><strong>3.47</strong></td>
<td><strong>280</strong></td>
</tr>
</tbody>
</table>

**Table 2: 30 random no-op starts.**
Thank you !