Rainbow: Combining Improvements in Deep Reinforcement Learning

Hessel et al. AAAI (2018)

CS 330 Student Presentation
Motivation

- Deep Q-Network successfully combines deep learning with reinforcement learning.
- Many extensions have been made to improve DQN:
  1. Double DQN
  2. Prioritized Replay
  3. Dueling Architecture
  4. Multi-step Learning
  5. Distributional RL
  6. Noisy Net
- Which of these improvements are complementary and can be fruitfully combined? → Rainbow
- Which extensions contribute the most in the “ensemble”? → Ablation study.
Background: Deep Q-Network

- CNN used to represent value function $q(s, a)$; It learns action values for each action given state (raw image pixels) as inputs.

- Agent uses an $\epsilon$-greedy policy and appends transition tuple $(S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1})$ to a replay buffer.

- It updates CNN parameters ($\theta$) to minimize:

  $$L_\theta = (R_{t+1} + \gamma \max_{a'} q_{\theta'}(S_{t+1}, a') - q_\theta (S_t, A_t))^2$$

  learning target

- The time step $t$ above is a randomly picked transition tuple from the replay buffer

- Parameter vector $\theta'$ which parameterizes the target network above represents a periodic copy of the online network parameterized by $\theta$
Double Q-learning

• Q-learning is vulnerable to overestimation bias because of a positive bias that results from using the maximum value as approximation for the maximum expected value.

• Double Q-learning aims to correct this positive bias by decoupling the selection of the action from its evaluation.

• The new loss function becomes:

\[
L_{\theta} = \left( R_{t+1} + \gamma \ q_{\theta'}(S_{t+1}, \ \text{argmax}_{a'} q_{\theta}(S_t, a')) - q_{\theta}(S_t, A_t) \right)^2
\]

• The above change results in sometimes underestimating the maximum expected value thereby reducing the overestimation bias on average.
Prioritized Replay

• Instead of uniform sampling from the replay buffer of DQN, develop a prioritized sampling approach that samples those transitions which might aid most in learning with higher probability

• Sample transitions according to a probability $p_t$ which is proportional to last encountered absolute TD error:

$$p_t \propto |R_{t+1} + \gamma \max_{a'} q_{\theta'}(S_{t+1}, a') - q_{\theta}(S_t, A_t)|^w$$

• This scheme prioritizes sampling of more recent transitions added to the replay buffer
Dueling Networks

- **Two computation streams:** valuation stream ($v_\eta$) and advantage stream ($a_\psi$)
- Both value and action streams take feature representation from a common CNN encoder ($f_\varepsilon$)
- Streams are joined by factorizing action values as follows:

\[ q_\theta(s, a) = v_\eta(f_\varepsilon(s)) + a_\psi(f_\varepsilon(s), a) - \frac{\sum_{a'} a_\psi(f_\varepsilon(s), a')}{N_{actions}} \]

- $\varepsilon$ are parameters of the shared encoder
- $\eta$ are parameters of the value stream
- $\psi$ are parameters of the action stream
- $\theta = \{\eta, \varepsilon, \psi\}$
Multi-step Learning

• Instead of a single step (greedy) approach, implement the bootstrapping after continuing up to some $n$ steps whose discounted reward is:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$$

• Now the standard DQN equation becomes:

$$\left( R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\theta'}(S_{t+n}, a') - q_{\theta}(S_t, A_t) \right)^2$$

• Authors noted that multi-step approach leads to faster learning where $n$ is a tunable hyperparameter
Distributional RL

• Instead of learning expected return $q(s, a)$, learn to estimate the distribution of returns whose expectation is $q(s, a)$.

• Consider a vector $z$ s.t.

\[ z^i = v_{\text{min}} + (i - 1) \frac{v_{\text{max}} - v_{\text{min}}}{N_{\text{atoms}} - 1}; \quad i \in \{1, \ldots, N_{\text{atoms}}\}, \quad N_{\text{atoms}} \in \mathbb{N}^+ \]

$z$ is the support of the return probability distribution (a categorical distribution)

• We want to learn a network with parameters $\theta$ which parameterizes the distribution $d_t \equiv (z, p_{\theta}(S_t, A_t))$, where $p_{\theta}(S_t, A_t)$ is the probability mass on each atom $i$ and we want to make $d_t$ close to the target distribution $d'_t \equiv (R_{t+1} + \gamma_{t+1} z, \ p_{\theta'}(S_{t+1}, a'_{t+1}))$

• KL divergence loss is used such that we want to minimize

\[
D_{KL}(\Phi_z d'_t || d_t)
\]

$\Phi_z$ is an L2-projection of the target distribution $d'_t$ onto the fixed/same support
Distributional RL

• Generate target distribution [1]:

Noisy Nets

• Replace linear layer with a **noisy linear layer** which combines deterministic and noisy streams

\[ y = (b + W x) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^w) x) \]

• Network should be able to learn to ignore the noisy stream after sufficient training but does so at different rates in different parts of the state-space, thereby allowing conditional exploration

• Instead of $\epsilon$-greedy policy, Noisy Nets inject **noise in the parametric space** for exploration.
Rainbow: The Integrated Agent

- Use **Distributional RL** to estimate return distribution rather than expected returns
- Replace 1-step distributional loss with **multi-step variant**
  
  This corresponds to a target distribution $d_t^{(n)} \equiv (R_{t+1}^{(n)} + \gamma_{t+1}^{(n)} z, p_{\theta'}(S_{t+n}, a_{t+n}^*))$ with the KL-loss as $D_{KL}(\Phi_z d_t^{(n)} || d_t)$
- Combine the above distributional loss with multi-step **double Q-learning**
- Incorporate the concept of **Prioritized replay** by sampling transitions which are prioritized by the KL loss.
- Network architecture is a **dueling architecture** adapted for use with return distributions
  
  For each atom $z^i$, the corresponding probability mass $p^i_\theta(s, a)$ is calculated as below by using a dueling architecture:

  $$p^i_\theta(s, a) \propto \exp(v^i_\eta(f^i_\epsilon(s))) + a^i_\psi(f^i_\epsilon(s), a) - \frac{\sum_{a'} a'^i_\psi(f^i_\epsilon(s), a')}{N_{actions}}$$

- Finally incorporate noisy streams by replacing all linear layers with their noisy equivalent
Rainbow: overall performance
Rainbow: Ablation Study

- **Prioritized replay** and **multi-step learning** were the two most crucial components of Rainbow.
- Removal of either hurts early performance but removal of multi-step even decreased final performance.
- Next most important is **Distributional Q-learning**.
- **Noisy Nets** generally outperform $\epsilon$-greedy approaches but in some games they hurt slightly.
- **Dueling networks don’t seem** to make much difference.
- Double Q-learning also does not help much and at times might even hurt performance.
- However, double Q-learning might become more important if a wider support range (rewards were clipped to [-10, +10] for these experiments) is used wherein overestimation bias might become more pronounced and the overestimation bias correction that double Q-learning provides might become more necessary.

Figure 3: Median human-normalized performance across 57 Atari games, as a function of time. We compare our integrated agent (rainbow-colored) to DQN (gray) and to six different ablations (dashed lines). Curves are smoothed with a moving average over 5 points.
Rainbow: Ablation + Performance by game type

Figure 2: Each plot shows, for several agents, the number of games where they have achieved at least a given fraction of human performance, as a function of time. From left to right we consider the 20%, 50%, 100%, 200% and 500% thresholds. On the first row we compare Rainbow to the baselines. On the second row we compare Rainbow to its ablations.
Rainbow: Experimental Results

- Rainbow performance significantly exceeds all competitor models on both data efficiency and overall performance.
- Rainbow outperforms other agents at all levels of performance: Rainbow agent improves scores on games where the baseline agents were already competitive, and also improves in games where baseline agents are still far from human performance.

<table>
<thead>
<tr>
<th>Agent</th>
<th>no-ops</th>
<th>human starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>79%</td>
<td>68%</td>
</tr>
<tr>
<td>DDQN (*)</td>
<td>117%</td>
<td>110%</td>
</tr>
<tr>
<td>Prioritized DDQN (*)</td>
<td>140%</td>
<td>128%</td>
</tr>
<tr>
<td>Dueling DDQN (*)</td>
<td>151%</td>
<td>117%</td>
</tr>
<tr>
<td>A3C (*)</td>
<td>-</td>
<td>116%</td>
</tr>
<tr>
<td>Noisy DQN</td>
<td>118%</td>
<td>102%</td>
</tr>
<tr>
<td>Distributional DQN</td>
<td>164%</td>
<td>125%</td>
</tr>
<tr>
<td>Rainbow</td>
<td>223%</td>
<td>153%</td>
</tr>
</tbody>
</table>

- No-ops start: episodes initialized with a random number (up to 30) of no-op actions
- Human start: episodes are initialized with points randomly sampled from the initial portion of human expert trajectories
- Difference between the two regimes indicates the extent to which the agent has over-fit to its own trajectories.
Rainbow: Advantages

• Good integration of various SOTA value based RL methods at the time
• Thorough ablation study provided for how different extensions contribute to the end result and performance and also their reasonings
• Very useful study for practitioners trying to sift through this quickly developing field as to what is important, how to combine and ablation study for sensitivity to different extensions of deep Q-learning
Rainbow: Disadvantages

- Ablation study **only considers elimination of single improvements**; this does not provide information about synergistic relationships between the improvements.
- Evaluation **only on 57 Atari 2600 games**, not other types of games/environments and also other non-game tasks.
- Experiments for Rainbow were done with Adam while older algorithms were done with others like RMSProp.
- Not clear how much of performance could be attributed to the combination approach versus others like better feature extractor (neural network architecture) or more robust hyperparameter tuning.
- No comparison with SOTA policy based or actor-critic approaches at the time.
- Source code from original authors not provided (to best of our knowledge) which clouds some of the implementation details and **hinders reproducibility**.
- **Computationally intensive**. Authors mention it takes 10 days for 200M frames (full run) on 1 GPU so for 57 games that would mean 570 days on a single GPU.