Rainbow - Combining Improvements in Deep Reinforcement Learning

IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures

Mohan Zhang
Problem settings

• The Markov Decision Process \(<S, A, T, r, \gamma>\)
• S states
• A actions
• \(T(s, a, s') = P[S_{t+1} = s'|S_t = s, A_t = a]\) (stochastic) transaction func
• \(r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]\) reward
• \(\gamma_t\) discount factor at time t
• Discounted return \(G_t = \sum_{k=0}^{\infty} \gamma^{(k)}_t R_{t+k+1}\)
• Discount factor \(\gamma^{(k)}_t = \prod_{i=1}^{k} \gamma_{t+i}\)
Tons of tricks in a nutshell! Ready?

THERE'S TOO MANY OF THEM
Value-based RL

- $v^\pi(s) = \mathbb{E}[G_t|S_t = s]$ or $Q^\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

- Then, with the value (or Q value) as a proxy, we could derive the policy $\pi$ with $\epsilon$-greedy argmax. (take max value action with probability $1 - \epsilon$ or uniformly from action space $A$ with probability $\epsilon$)
DQN

• A deep Q network (DQN) is a multi-layered neural network that for a given state $s$ outputs a vector of action values $Q(s, \cdot ; \theta)$, where $\theta$ are the parameters of the network.

• **target network**: its parameters ($\theta^-$) are copied every episode from the online network ($\theta$) to make training more stable.

$$Y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-).$$  (3)

• **reply buffers** (Experience reply): transitions, rewards and actions are stored for some time and sampled uniformly from this memory bank to update the network. This is to prevent our DNN to overfit the current episode.
Double Q-Learning

• Two separate value functions (DNNs in our case)
• Pick a batch of experience, then assign each experience randomly to one of the DNN to update it. After this, we get two set of params $\theta$ and $\theta'$
• For each update, one set of DNN is used to determine the action greedily, the other is used to determine the Q value.

\[
Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t). \quad (2)
\]

\[
Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta'_t). \quad (4)
\]

\[
Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-). \quad (4)
\]
From Double Q-Learning to Prioritized Replay

• For SGD, we used this to measure the temporal-difference (TD) error:

\[ \Delta = R_{t+1} + \gamma_{t+1} \arg\max_{a'} Q_{\theta^-}(S_{t+1}, a') - Q_{\theta}(S_t, A_t) \]

• We perform gradient descent over \( \theta \), then update \( \theta^- \) in the beginning of every episode.
Backup vanilla Q-Learning

\[ Q_\pi(s, a) \equiv \mathbb{E} [R_1 + \gamma R_2 + \ldots | S_0 = s, A_0 = a, \pi] \]
\[ \equiv \mathbb{E} [ R_1 + \gamma Q_\pi(s_{t+1}, a')] \]

- \( \pi \): policy
- \( \gamma \): discount factor
- R: reward
- S: state
- A: action
- Then, the optimal Q value \( Q_*(s, a) = \max_\pi Q_\pi(s, a) \)
Backup vanilla Q-Learning

• The optimal Q value can be learned from Q Learning

• In most cases, we cannot go over all action values in all states separately. So, we parametrize the Q value by $\theta$: $Q(s, a; \theta_t)$, which can be updated with SGD:

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t). \quad (1)$$

$$Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t). \quad (2)$$

• $Y_t^Q$ represents the optimal Q value given best choice of $\theta$

• $\alpha$ is the learning rate
Prioritized Replay

- DQN samples uniformly from the replay buffer.
- We sample import *(with high expected learning progress)* transactions more frequently.
- Sample probability given the (traditional) experience \(<S_t, A_t, R_t, S_{t+1}>\)

\[
p_t \propto \left| R_t + \gamma_t \max_{a'} q_{\theta}(S_{t+1}, a') - q_{\theta}(S_t, A_t) \right|^{\omega}
\]

- \(\omega\) is a hyper-parameter that determines the shape of the distribution.
- Note that stochastic transitions might also be favored, even when there is little left to learn about them, in order to avoid overfitting.
Dueling Networks

- Basically, add another network to evaluate action advantages.
- We evaluate “goodness” (on the edge or not) of a state \( s \) and advantage of choosing an action \( a \) (turn left or right).

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

- \( v_\eta \): value of state. \( \eta \) is the params of such value stream.
- \( a_\psi \): value of action advantage. \( \psi \) is the params of such advantage stream.
- \( f_\xi \) is the shared convolutional encoder (network)
Multi-step Learning

• **truncated** n-step return from a given state $S_t$:
  $$ R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} $$

• Then, the multi-step variant of DQN is then defined by minimizing the alternative loss (same thing as before, just changed $R_t$ to be $R_t^{(n)}$)
  $$(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_\theta(S_{t+n}, a') - q_\theta(S_t, A_t))^2$$

• Multi-step targets with suitably tuned n often lead to faster learning
Distributional RL

• learn to approximate the distribution of returns instead of the expected return.
• Maximize over the expected sum of future rewards.
• New Bellman: $V_\pi(x) \equiv \mathbb{E}_{P^\pi} [\sum_t \gamma^t R(x_t) | x_0 = x] = \mathbb{E} R(x) + \mathbb{E}_{x' \sim P^\pi} V_\pi(x')$
• Future expectation makes modeling even more complex! We use a hidden variable $z$ to model the value distribution:
  - $V_\pi(x) = \mathbb{E} Z_\pi(x) = \mathbb{E} [R(x) + \gamma Z_\pi(x')]$, where $x' \sim P^\pi(\cdot | x)$
• Discrete distributions C51 to measure. Simply replace the Q-output in DQN to a softmax over 51 probabilities.(more bins, better performance!)
Distributional RL

• The equations for a distributional variance of Q-learning: constructing a new support $d_t$ by minimizing the KL divergence between $d_t$ and target $d_t'$.

\[ d'_t \equiv (R_{t+1} + \gamma_{t+1} z, \, p_{\tilde{\theta}}(S_{t+1}, \bar{a}_{t+1}^*)) \],
\[ D_{\text{KL}}(\Phi_z d'_t \| d_t) \].
Noisy Nets

• where many actions must be executed to collect the first reward (Montezuma’s Revenge), what do we do?

• Add noise for better exploration!

\[ y = (b + Wx) + (b_{noisy} \odot \varepsilon^b + (W_{noisy} \odot \varepsilon^w)x), \quad (4) \]

• Over time, the network can learn to ignore the noisy stream at different rates in different parts of the state space

• self-annealing
Now, group together!
Recap

\begin{align}
(R_{t+1} + \gamma_{t+1} \max_{c'} q_{\theta}(S_{t+1}, a') - q_{\theta}(S_t, A_t))^2,
\end{align}

(1)

\begin{align}
R_t^{(n)} &\equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}.
\end{align}

(2)

d'_t \equiv (R_{t+1} + \gamma_{t+1} z, \ p_{\theta}(S_{t+1}, \bar{a}_{t+1}^*)) , \\
 D_{KL}(\Phi z d' \| d_t).
\end{align}

(3)

\begin{align}
 y = (b + W x) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^w) x).
\end{align}

(4)
Multi-step Learning

\[ R^{(n)}_t = \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} \]  

(2) \[ d'_t = (R_{t+1} + \gamma_{t+1} z, \ p_{\theta}(S_{t+1}, a^*_{t+1})) \]

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

Prioritized reply

\[ d^{(n)}_t = (R^{(n)}_t + \gamma^{(n)}_t z, \ p_{\theta}(S_{t+n}, a^*_{t+n})) \]

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

Target net

\[ Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t, \theta^-)) \]

Double DQN

Distibutional RL

Final target KL to minimize

\[ p_t \propto (D_{KL}(\Phi_z d^{(n)}_t || d_t))^\omega \]

Dueling network

\[ p_t \propto R_t + \gamma_t \max_{a'} q_{\theta}(S_{t+1}, a') - q_{\theta}(S_t, A_t) \]

Prioritized reply
Experiments:

• “All Rainbow’s components have a number of hyper-parameters. The combinatorial space of hyper-parameters is too large for an exhaustive search, therefore we have performed limited tuning.”

<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>185%</td>
<td>125%</td>
</tr>
<tr>
<td>Rainbow</td>
<td>231%</td>
<td>153%</td>
</tr>
</tbody>
</table>
Experiments:

• Double DQN is redundant?
• Is it just useless or the functionality is shadowed by the combination of other tricks?
• Pros:
All tricks together, SOTA performance!
A good base to construct your other algorithms on

• Cons:
Hard to tune, hard to implement
No clue how to make it more efficient
IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures

(a) Batched A2C (sync step.)

(b) Batched A2C (sync traj.)

(c) IMPALA
IMPALA
V trace correction

\[ v_s \overset{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left( \prod_{i=s}^{t-1} c_i \right) \delta_t V, \quad (1) \]

where \( \delta_t V \overset{\text{def}}{=} \rho_t \left( r_t + \gamma V(x_{t+1}) - V(x_t) \right) \)

\[ c_i \overset{\text{def}}{=} \min \left( \bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)} \right) \]

if \( \mu > \pi \)
My Questions:

• Where were they from?

\[
(R_{t+1} + \gamma_{t+1} \max_{c'} q_\theta(S_{t+1}, a') - q_\theta(S_t, A_t))^2, \quad (1)
\]

\[
R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}. \quad (2)
\]

\[
d_t' \equiv (R_{t+1} + \gamma_{t+1} z, \ p_\theta(S_{t+1}, \bar{a}_{t+1})),
\]

\[
D_{KL}(\Phi_z d_t' || d_t). \quad (3)
\]

\[
y = (b + Wx) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^w)x), \quad (4)
\]
• What is the most important contribution of IMPALA? (hint: distributed)