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, γ>
- 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
- γ_t discount factor at time t
- Discounted return $G_t = \sum_{k=0}^{\infty} \gamma_t^{(k)} R_{t+k+1}$
- Discount factor $\gamma_t^{(k)} = \prod_{i=1}^{k} \gamma_{t+i}$

Tons of tricks in a nutshell! Ready?

Value-based RL

- $\mathrm{v}^{\pi}(\mathrm{s}) = \mathbb{E}[G_t | S_t = s]$ 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 π with ϵ-greedy argmax. (take max value action with probability 1- ϵ or uniformly from action space A with probability ϵ)

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 θ are the parameters of the network.
- **target network**: its parameters (θ^-) are copied every episode from the online network (θ) to make training more stable.

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-)\,. \qquad (3)$$

- **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 θ and θ'
- For each update, one set of DNN is used to determine the action greedily, the other is used to determine the Q value.

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t) \,. \qquad (2)$$

$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t) \,. \quad (4)$$

$$Y_t^{\mathrm{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^{-}) \,.$$

From Double Q-Learning to Prioritized Replay

- For SGD, we used this to measure the *temporal-difference (TD) error:*
- $\Delta = R_{t+1} + \gamma_{t+1} argmax_{a'} Q_{\theta^-}(S_{t+1}, a') - Q_\theta(S_t, A_t)$
- We perform gradient descent over θ, then update θ^- in the beginning of every episode.

Backup vanilla Q-Learning

$$Q_\pi(s, a) \equiv \mathbb{E}\left[R_1 + \gamma R_2 + \ldots \mid S_0 = s, A_0 = a, \pi\right]$$

$$\equiv \mathbb{E}\left[R_1 + \gamma Q_\pi(s_{t+1}, a')\right]$$

- π: policy
- γ: 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 θ: $Q(s, a; \theta_t)$, which can be updated with SGD:
- $$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^{\mathrm{Q}} - Q(S_t, A_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(S_t, A_t; \boldsymbol{\theta}_t)\,. \quad (1)$$
- $$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)\,. \quad (2)$$
- Y_t^Q represents the optimal Q value given best choice of θ
- α 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}>

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

- ω 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

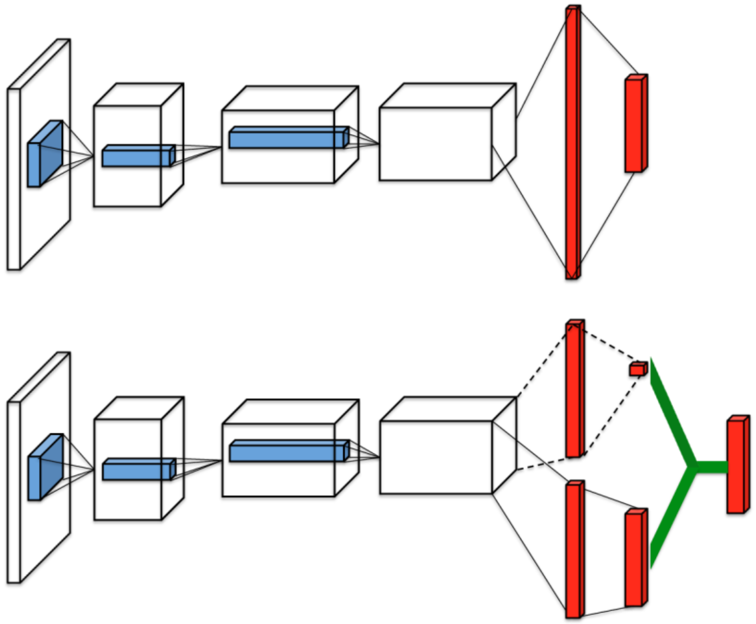
- Basically, add another network to evaluate action advantages.
- We evaluate "goodness" (on the edge or not) of a state s and advantage of choosing an action a (turn left or right).

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

- v_η: value of state. η is the params of such value stream.
- a_ψ: value of action advantage. ψ is the params of such

advantage stream.

- f_ξ is the shared convolutional encoder (network)



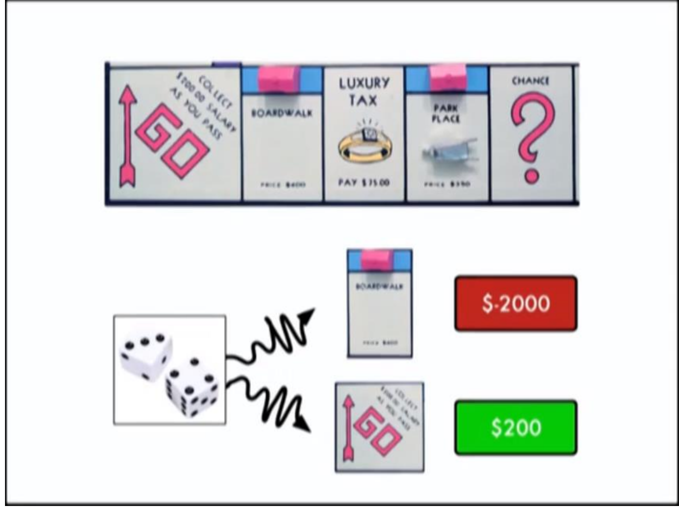
Multi-step Learning

- **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 $R_t^{(n)}$

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

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

Distributional RL



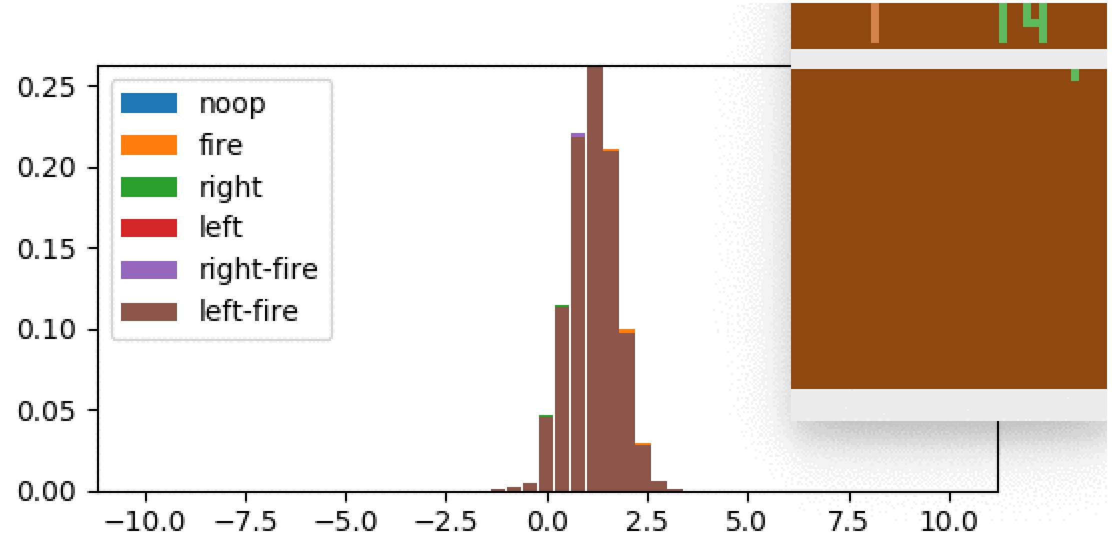
- learn to approximate the **distribution of returns** instead of the expected return.
- Maximize over the *expected* sum of future rewards.
- New Bellman: $V^{\pi}(x) \equiv \mathbb{E}_{P^{\pi}}[\sum_t \gamma^t R(x_t) | x_0 = x] = \mathbb{E}R(x) + \mathbb{E}_{x' \sim P^{\pi}} V^{\pi}(x')$
- Future expectation 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 $x' \sim \mathrm{P}^{\pi}(\cdot\, | x)$
- Discrete distributions C51 to measure. Simply replace the Q-output in DQN to a softmax over 51 probabilities.(more bins, better performance!)

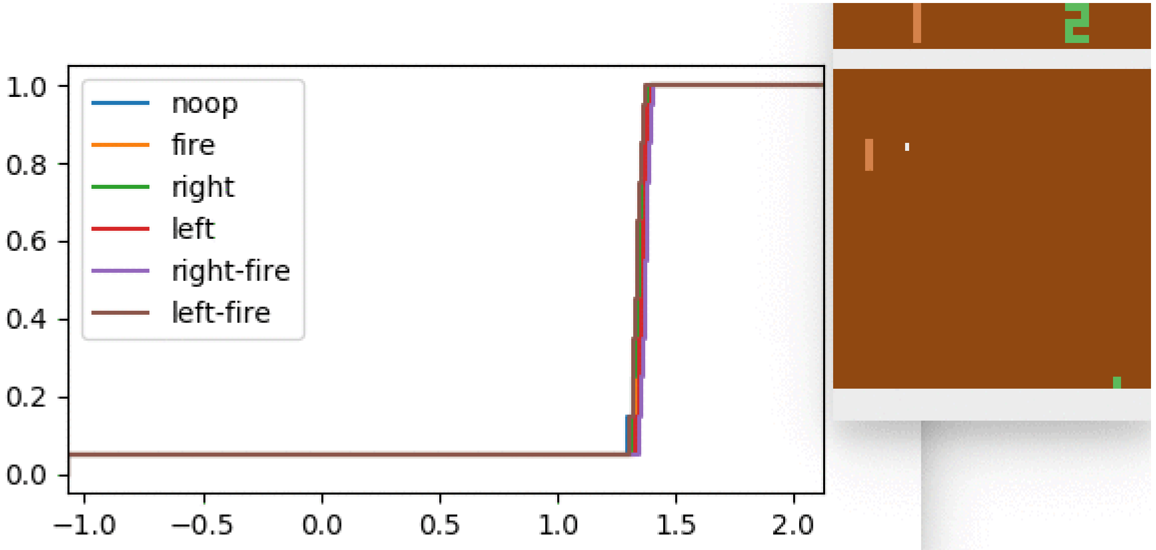
Distributional RL

- 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'.

$$d'_t \equiv (R_{t+1} + \gamma_{t+1} \boldsymbol{z},\ \boldsymbol{p}_{\overline{\theta}}(S_{t+1}, \overline{a}^*_{t+1})),$$

$$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d'_t || d_t)\ .$$



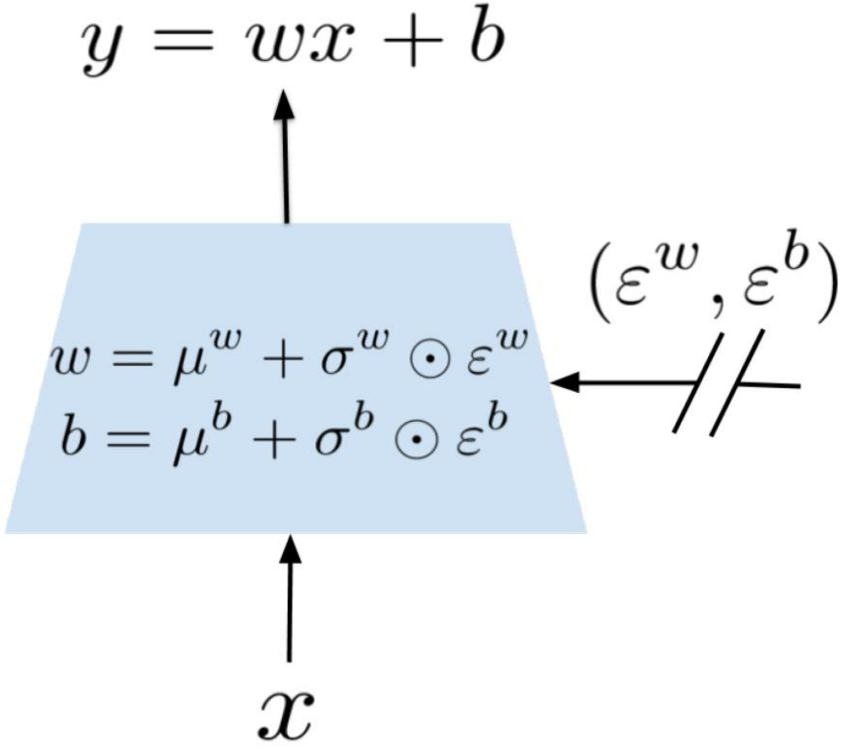


Noisy Nets

- where many actions must be executed to collect the first reward (Montezuma's Revenge), what do we do?
- Add noise for better exploration!

$$\boldsymbol{y} = (\boldsymbol{b} + \mathbf{W}\boldsymbol{x}) + (\boldsymbol{b}_{noisy} \odot \epsilon^b + (\mathbf{W}_{noisy} \odot \epsilon^w)\boldsymbol{x}), \quad (4)$$

- 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

$$\left(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t)\right)^2 , \qquad (1)$$

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} \, . \qquad (2)$$

$$d'_t \equiv (R_{t+1} + \gamma_{t+1} \boldsymbol{z}, \;\; \boldsymbol{p}_{\overline{\theta}}(S_{t+1}, \overline{a}^*_{t+1})),$$

$$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d'_t || d_t) \, . \qquad (3)$$

$$\boldsymbol{y} = (\boldsymbol{b} + \mathbf{W}\boldsymbol{x}) + (\boldsymbol{b}_{noisy} \odot \epsilon^b + (\mathbf{W}_{noisy} \odot \epsilon^w)\boldsymbol{x}), \quad (4)$$

Multi-step Learning

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} \,. \qquad (2)$$

Distributional RL

$$d'_t \equiv (R_{t+1} + \gamma_{t+1}\boldsymbol{z},\ \boldsymbol{p}_{\overline{\theta}}(S_{t+1}, \overline{a}^*_{t+1})),$$
$$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d'_t || d_t) \,. \qquad (3)$$

$$d_t^{(n)} = (R_t^{(n)} + \gamma_t^{(n)} \boldsymbol{z},\quad \boldsymbol{p}_{\overline{\theta}}(S_{t+n}, a^*_{t+n}))$$

Target net

$$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t)$$

$$Y_t^{\mathrm{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-) \,.$$

Double DQN

$$p_t \propto \left(D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t) \right)^\omega$$

Final target KL to minimize

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

Dueling network

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

Prioritized reply

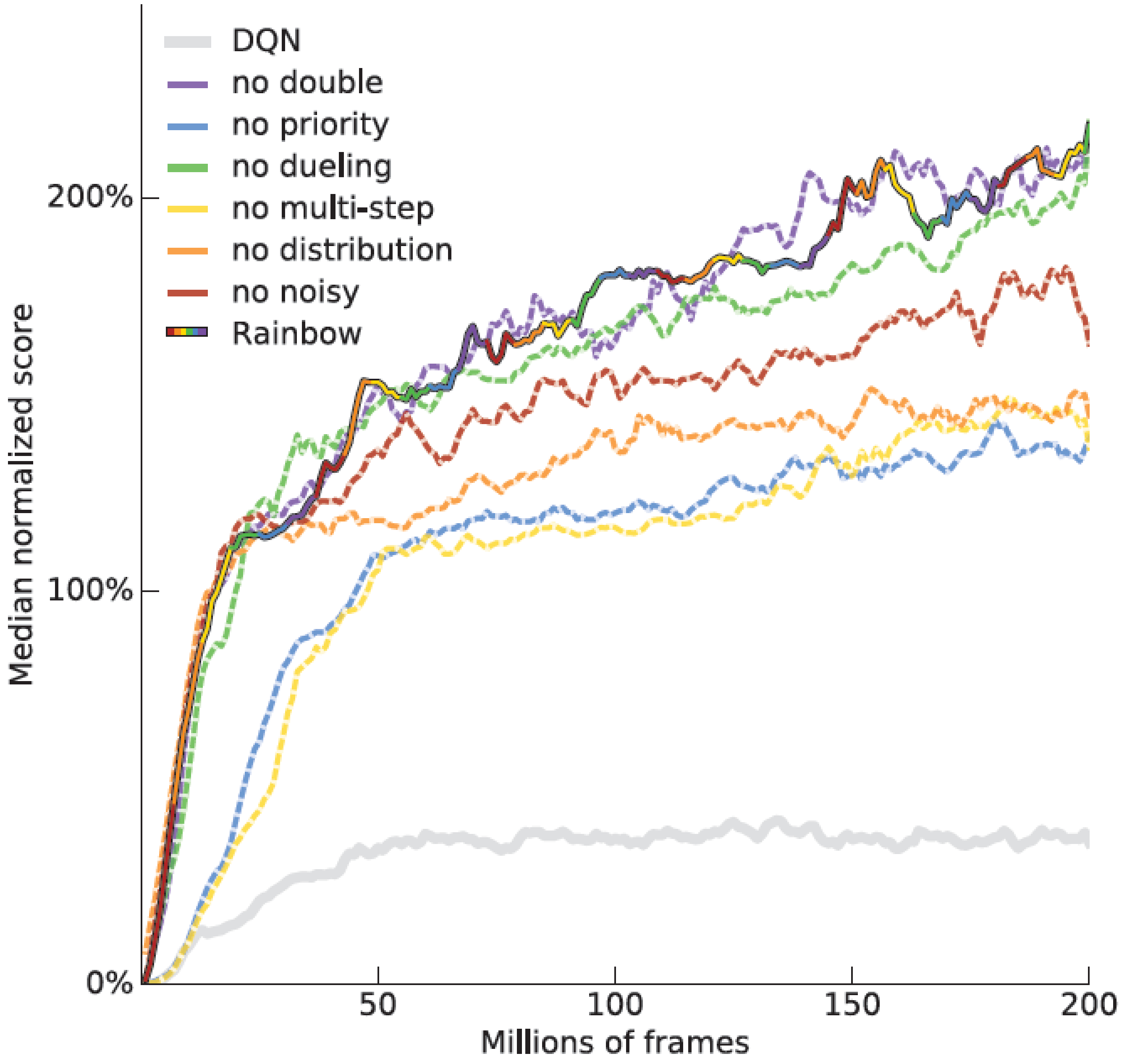
Experiments:

- "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."

Agent	no-ops	human starts
DQN	79%	68%
DDQN (*)	117%	110%
Prioritized DDQN (*)	140%	128%
Dueling DDQN (*)	151%	117%
A3C (*)	-	116%
Noisy DQN	118%	102%
Distributional DQN	185%	125%
Rainbow	231%	153%

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!

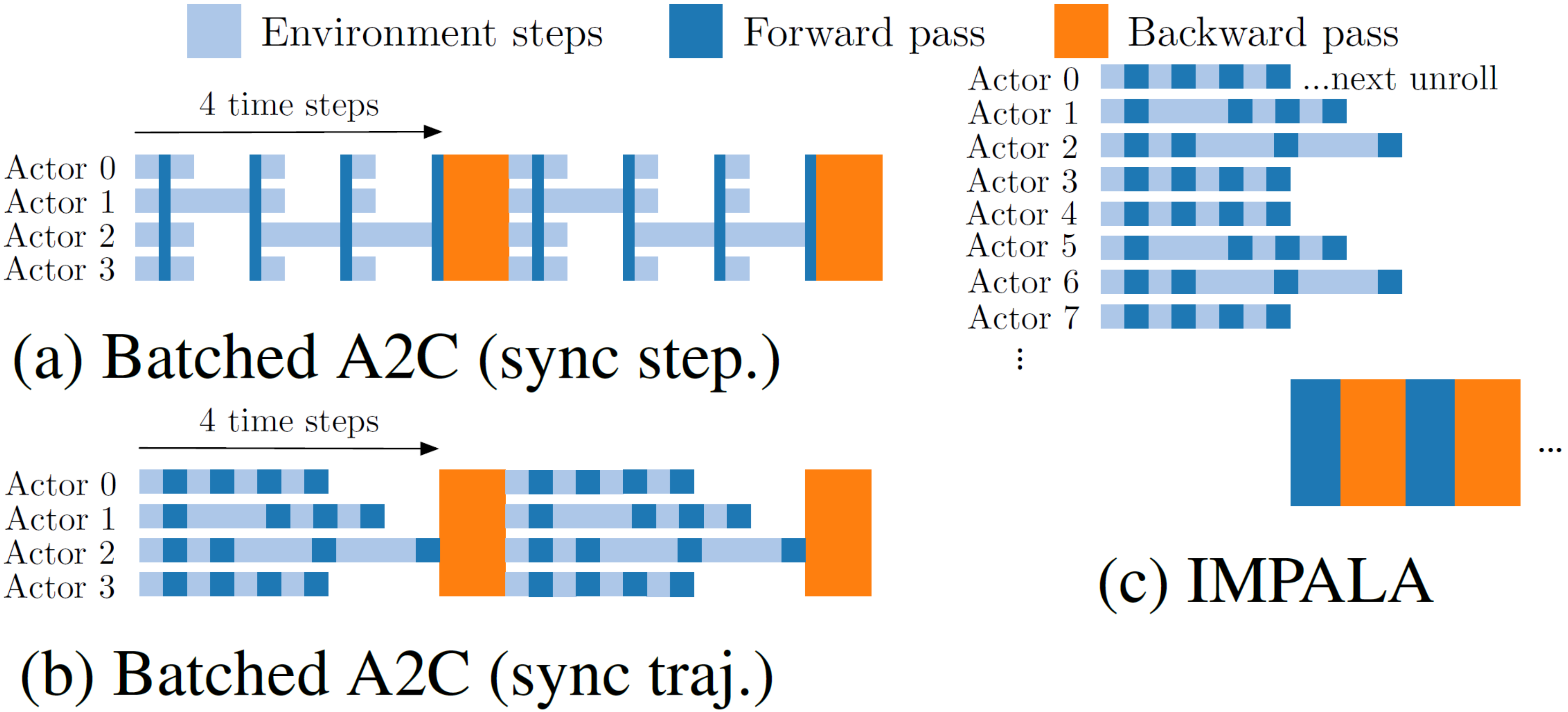
A good base to construct your other algorithms on

- Cons:

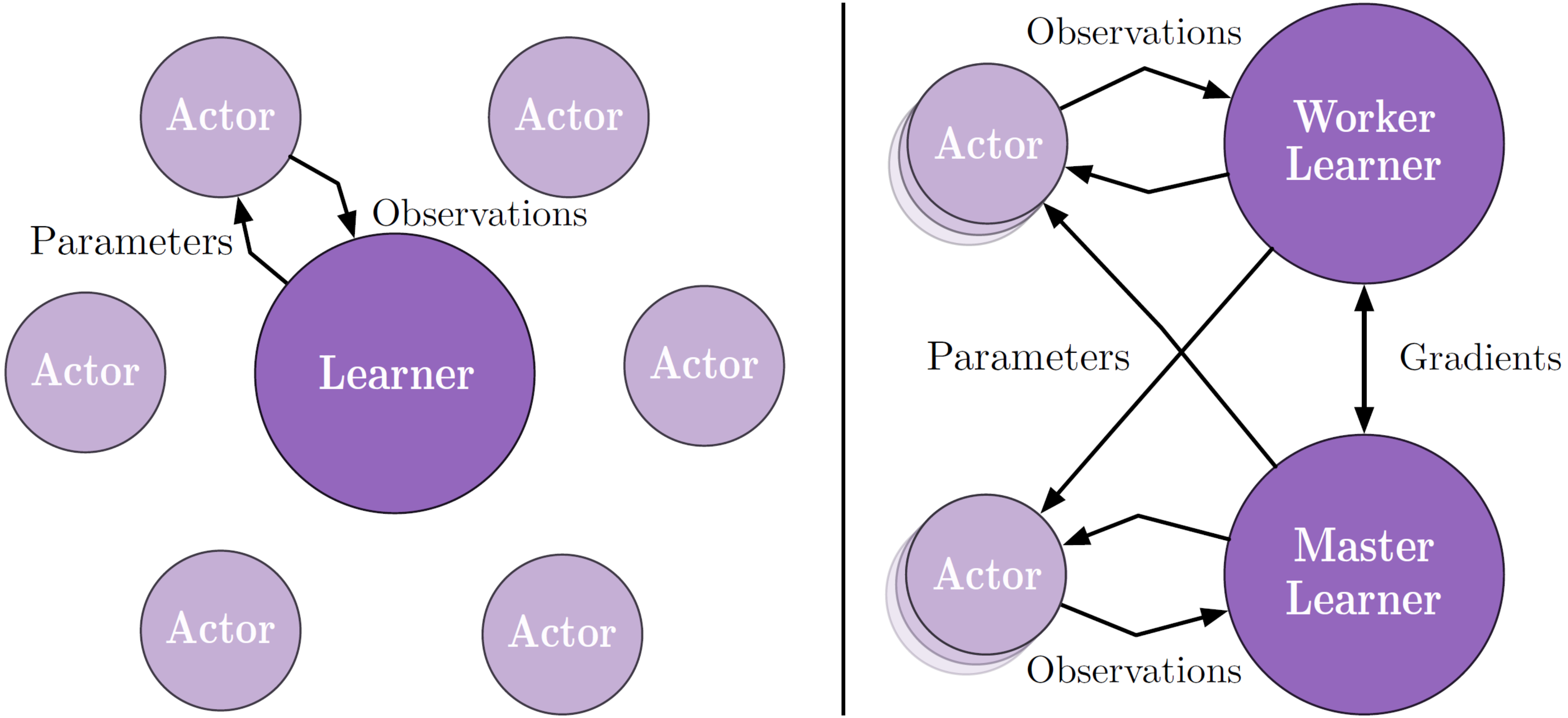
Hard to tune, hard to implement

No clue how to make it more efficient

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



IMPALA



Actor
Actor
Parameters
Observations
Actor
Learner
Actor
Actor
Actor
Observations
Actor
Worker Learner
Parameters
Gradients
Actor
Master Learner
Observations

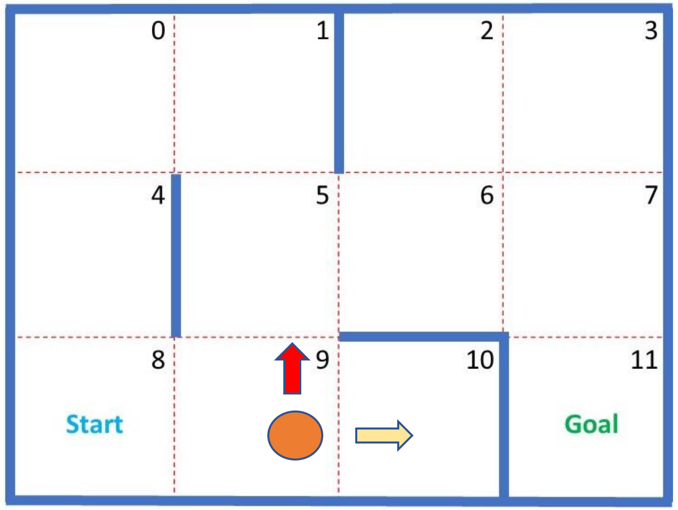
V trace correction

$$v_s \stackrel{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \Big(\prod_{i=s}^{t-1} c_i \Big) \delta_t V, \quad (1)$$

where $\delta_t V \stackrel{\text{def}}{=} \rho_t \big(r_t + \gamma V(x_{t+1}) - V(x_t)\big)$

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

if $\mu > \pi$



My Questions:

- Where were they from?

$$(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t))^2 \ , \qquad (1)$$

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} \, . \qquad (2)$$

$$d'_t \equiv (R_{t+1} + \gamma_{t+1}\boldsymbol{z}, \ \ \boldsymbol{p}_{\overline{\theta}}(S_{t+1}, \overline{a}^*_{t+1})),$$

$$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d'_t || d_t) \, . \qquad (3)$$

$$\boldsymbol{y} = (\boldsymbol{b} + \mathbf{W}\boldsymbol{x}) + (\boldsymbol{b}_{noisy} \odot \epsilon^b + (\mathbf{W}_{noisy} \odot \epsilon^w)\boldsymbol{x}), \quad (4)$$

- What is the most important contribution of IMPALA?

(hint: distributed)