Deep Q Learning[1]

[1] Sections 4.3 and Chapter 5: Yang&Ying

Issues of nonlinear function approximation for reinforcement learning

- No theoretical guarantees on optimality and convergence
- Sometimes, unstable during training

Next we will introduce a few ideas that make deep reinforcement learning more stable.

Replay Buffer and Experience Replay

Pretend we have a supervised learning problem

$$\frac{1}{2}\min_{w}\sum_{k}\Big(y_k-\hat{Q}(x_k,u_k;w)\Big)^2,$$

where $y_k = r(x_k,u_k)+\alpha\max_v \hat{Q}(x_{k+1},v;w)$ is the label of sample (x_k,u_k) (we ignore the fact that y_k is a function of w like in linear function approximation.)

Replay Buffer and Experience Replay

Pretend we have a supervised learning problem

$$\frac{1}{2}\min_{w}\sum_{k}\Big(y_k-\hat{Q}(x_k,u_k;w)\Big)^2,$$

where $y_k = r(x_k,u_k)+\alpha\max_v \hat{Q}(x_{k+1},v;w)$ is the label of sample (x_k,u_k) (we ignore the fact that y_k is a function of w like in linear function approximation.)

Stochastic Gradient Descent (SGD)

$$w \leftarrow w+\beta_k(y_k-\hat{Q}(x_k,u_k;w))\nabla_w\hat{Q}(x_k,u_k;w).$$

Replay Buffer and Experience Replay

Stochastic Gradient Descent (SGD)

$$w \leftarrow w + \beta_k(y_k - \hat{Q}(x_k, u_k; w))\nabla_w \hat{Q}(x_k, u_k; w).$$

Deep neural networks trained with SGD have superior performance, assuming independently and identically distributed data samples $(y_k, (x_k, u_k))$.

Replay Buffer and Experience Replay

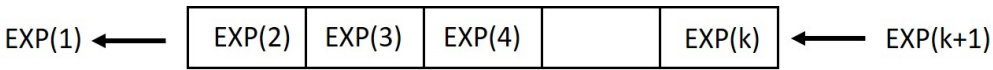
Stochastic Gradient Descent (SGD)

$$w \leftarrow w + \beta_k (y_k - \hat{Q}(x_k, u_k; w)) \nabla_w \hat{Q}(x_k, u_k; w).$$

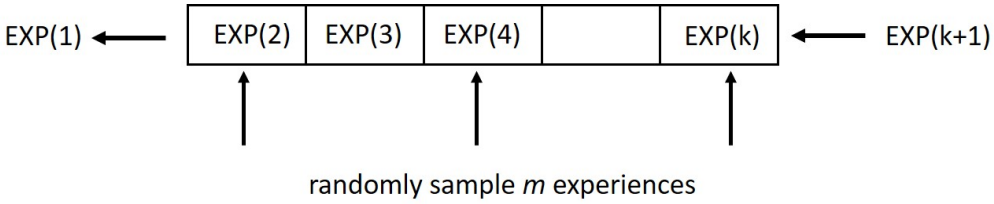
$$(y_1, (x_1, u_1)) \rightarrow (y_2, (x_2, u_2)) \rightarrow (y_3, (x_3, u_3)) \rightarrow \cdots$$

are highly correlated samples in reinforcement learning applications.

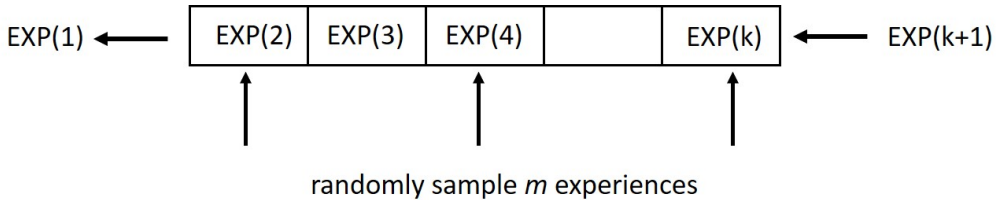
Replay Buffer



Replay Buffer and Experience Replay



Replay Buffer and Experience Replay



Minibatch SGD

$$w \leftarrow w + \beta_k \sum_{b=1}^{m} (y_{k,b} - \hat{Q}(x_{k,b}, u_{k,b}; w)) \nabla_w \hat{Q}(x_{k,b}, u_{k,b}; w).$$

Target Network

From the supervised learning perspective, the target value $y_{k,m} = r(x_{k,m}, u_{k,m}) + \alpha \max_v \hat{Q}(x_{k+1,m}, v; w)$ should be independent of w.

Target Network

From the supervised learning perspective, the target value $y_{k,m} = r(x_{k,m}, u_{k,m}) + \alpha \max_v \hat{Q}(x_{k+1,m}, v; w)$ should be independent of w.

Target Network Q_{target}

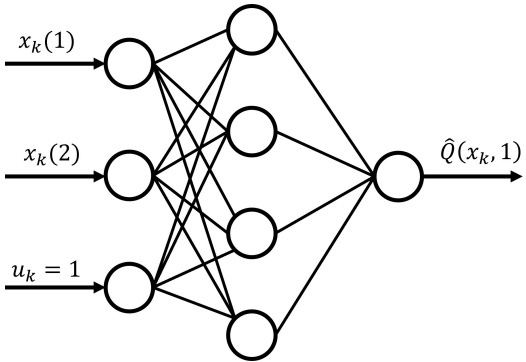
copy w to w_{target} once every n iterations and use Q_{target} to calculate $y_{k,m}$.

Deep Q-Network (DQN)

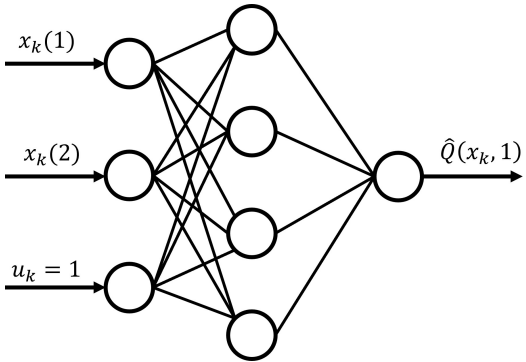
How to calculate $\sum_{b=1}^{m}(y_{k,b} - \hat{Q}(x_{k,b}, u_{k,b}; w))\nabla_w \hat{Q}(x_{k,b}, u_{k,b}; w)$ and update w?

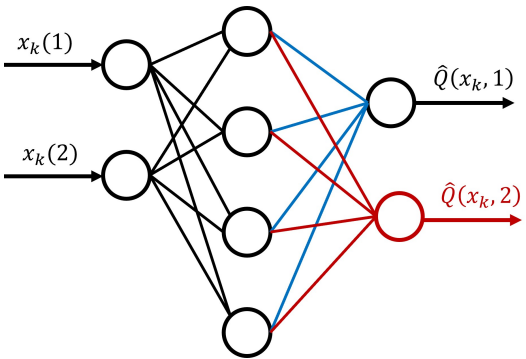
Deep Q-Learning in PyTorch

Deep Q-Network Architecture



Deep Q-Network Architecture

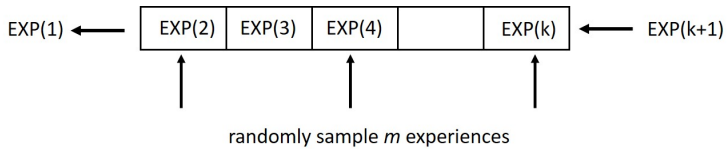




Couple the estimation of Q-values ($Q(x, u)$) of different actions u for the same state x.

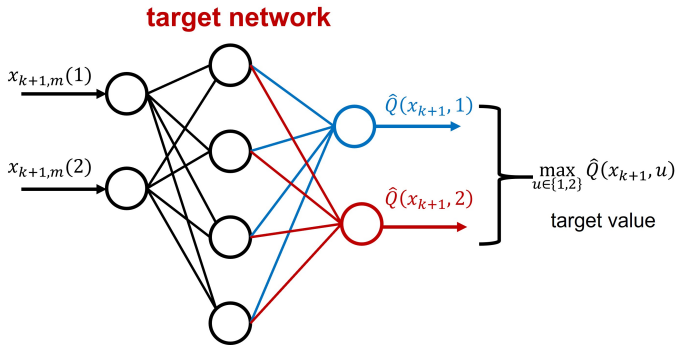
Deep-Q-Network (DQN) (Minh et al 2015) in PyTorch

- Sample a mini-batch $\{(x_{k,b}, u_{k,b}, r_{k,b}, x_{k+1,b})\}_{b=1,\cdots,m}$.
  random.sample(self.replay_buffer, batch_size)



Deep-Q-Network (DQN) (Minh et al 2015) in PyTorch

- Generate target-values $y_{k,m}$ using the target network
  next_state_values = target_net(next_states).max(1)[0]
  target_values= rewards+ gamma × next_state_values.detach()

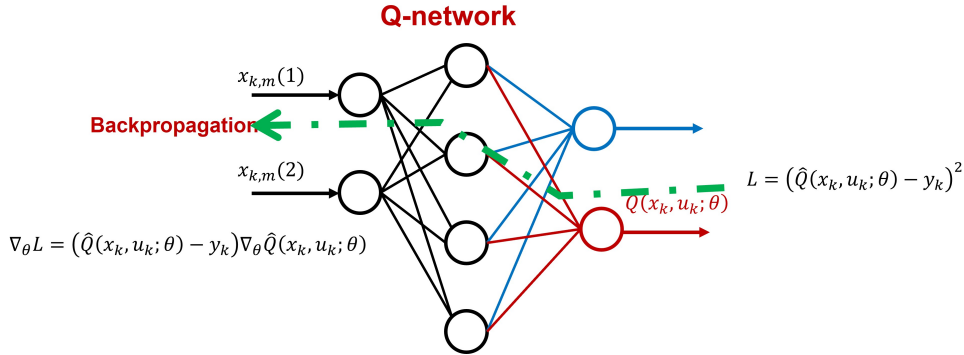


Deep-Q-Network (DQN) (Minh et al 2015) in PyTorch

- Define a loss function $L(w) = \sum_{b=1}^{m} \left(\hat{Q}(x_{k,b}, u_{k,b}; w) - y_{k,b} \right)^2$.
  loss = nn.MSELoss()(state_action_values, target_values)

Deep-Q-Network (DQN) (Minh et al 2015) in PyTorch

- Update $w \leftarrow w - \beta_k \nabla_w L(w)$
  optimizer.zerograd() loss.backward() optimizer.step()



Double Q-learning

Motivation

Q-learning overestimates action values under certain conditions, because of the maximization step over estimated action values.

$$\mathbb{E}\left[\max_v X_v\right] \geq \max_v \mathbb{E}\left[X_v\right].$$

Double Q-learning

Thrun and Schwartz (1993)

Assume under function approximation,

$$\tilde{Q}(i,u) = Q(i,u) + Y_{i,u}$$

where $Y_{i,u}$ is the noise and $\mathbb{E}[Y_{i,u}] = 0$ (zero mean).
Q-learning: given experience (i,u,j),

$$\tilde{Q}(i,u) \leftarrow r(i,u) + \alpha \max_{v} \tilde{Q}(j,v)$$

Double Q-learning

Estimation error from the noise

$$
\begin{aligned}
Z &= r(i,u) + \alpha \max_v \tilde{Q}(j,v) - \Big(r(i,u) + \alpha \max_v Q(j,v) \Big) \\
&= \alpha \Big(\max_v \tilde{Q}(j,v) - \max_v Q(j,v) \Big) \\
&= -\alpha \Big(\max_v Q(j,v) - \max_v \left(Q(j,v) + Y_{j,v} \right) \Big)
\end{aligned}
$$

Claim: $\mathbb{E}[Z] > 0$ even we have $\mathbb{E}[Y_{i,v}] = 0$.

Double Q-learning

Example

Consider the case

$$Q(j,u) = Q(j,v) \quad \forall u \neq v$$

i.e. Q-values are the same for all actions at the same state j. Assume $Y_{j,v} \sim \text{uniform}[-\epsilon, \epsilon]$ and independent across the actions.

Double Q-learning

Example

Consider the case

$$Q(j,u) = Q(j,v) \quad \forall u \neq v$$

i.e. Q-values are the same for all actions at the same state j. Assume $Y_{j,v} \sim \text{uniform}[-\epsilon, \epsilon]$ and independent across the actions.

$$\begin{aligned}\mathbb{E}[Z] &= \alpha \mathbb{E}\left[-Q(j,u) + Q(j,u) + \max_v Y_{j,v}\right] \\ &= \alpha \mathbb{E}\left[\max_v Y_{j,v}\right].\end{aligned}$$

Double Q-learning

Let n be the number of actions and $f(\cdot)$ be the PDF of $Y_{j,v}$. We have

$$\begin{aligned}\mathbb{E}[Z] &= \alpha \int_{-\infty}^{\infty} xnf(x) \left(\int_{-\infty}^{x} f(z)dz \right)^{n-1} dx \\ &= \alpha n \int_{-\epsilon}^{\epsilon} x \frac{1}{2\epsilon} \left(\frac{1}{2} + \frac{x}{2\epsilon} \right)^{n-1} dx\end{aligned}$$

Double Q-learning

Define $y = \frac{1}{2} + \frac{x}{2\epsilon}$, we have

$$\begin{aligned}
\mathbb{E}[Z] &= \alpha n \int_0^1 (2\epsilon y - \epsilon) y^{n-1} dy \\
&= \alpha n \epsilon \int_0^1 2y^n - y^{n-1} dy \\
&= \alpha n \epsilon \left(\frac{2}{n+1} - \frac{1}{n} \right) \\
&= \alpha \epsilon \frac{n-1}{n+1}.
\end{aligned}$$

Double Q-Learning (van Hasselt 2010)

Maintain two Q-functions, denoted by their parameters w and w'. For each sample, randomly pick one estimator to update

Estimator 1

$$w_{k+1} = w_k + \beta_k \delta_k \nabla_{w_k} Q(x_k, u_k; w_k) \quad \text{with}$$

$$\delta_k = r(x_k, u_k) + \alpha Q\left(x_{k+1}, \arg\max_v Q(x_{k+1}, v; w_k); w'_k\right) - Q(x_k, u_k; w_k).$$

Estimator 2

$$w'_{k+1} = w'_k + \beta_k \delta'_k \nabla_{w'_k} Q(x_k, u_k; w'_k) \quad \text{with}$$

$$\delta'_k = r(x_k, u_k) + \alpha Q\left(x_{k+1}, \arg\max_v Q(x_{k+1}, v; w'_k); w_k\right) - Q(x_k, u_k; w'_k).$$

Double Estimator is an Underestimator (Van Hasselt '2010)

Consider M random variables X_i. For each random variable X_i, we obtain a set of i.i.d. samples, denoted by $\mathcal{S}_i$.

Single estimator

$$\max_i \mathbb{E}[X_i] \approx \max_i \frac{1}{|\mathcal{S}_i|} \sum_{s \in \mathcal{S}_i} s$$

Double estimator

Split each $\mathcal{S}_i$ into $\mathcal{S}_i^{(1)}$ and $\mathcal{S}_i^{(2)}$ such that $\mathcal{S}_i^{(1)} \cup \mathcal{S}_i^{(2)} = \mathcal{S}_i$ and $\mathcal{S}_i^{(1)} \cap \mathcal{S}_i^{(2)} = \varnothing$.
Define

$$\mu_i^{(1)} = \frac{1}{\left|\mathcal{S}_i^{(1)}\right|} \sum_{s \in \mathcal{S}_i^{(1)}} s, \qquad \mu_i^{(2)} = \frac{1}{\left|\mathcal{S}_i^{(2)}\right|} \sum_{s \in \mathcal{S}_i^{(2)}} s.$$

- pick i^* such that $\mu_{i^*}^{(1)} = \max_i \mu_i^{(1)}$ and approximate $\max_i \mathbb{E}[X_i] = \mu_{i^*}^{(2)}$.

Cons

The double estimator to estimate $\max_i \mathbb{E}[X_i]$

Separate the maximizer and evaluation.

Theorem

The double estimator is an underestimator.

Proof

If i^* is the maximizer, then

$$\mathbb{E}\left[\mu_{i^*}^{(2)}\right] = \mathbb{E}[X_{i^*}] = \max_i \mathbb{E}[X_i].$$

If not, $\mathbb{E}[\mu_{i^*}^{(2)}] < \max_i \mathbb{E}[X_i]$. Therefore, in summary, we have

$$\mathbb{E}\left[\mu_{i^*}^{(2)}\right] \leq \max_i \mathbb{E}[X_i].$$

Q-Learning versus Double Q-Learning (Weng et al, 2020)

- Consider Q-Learning with learning rate $\beta_k = \frac{c}{k}$ and Double Q-Learning with learning rate $\beta_k = \frac{2c}{k}$.
- Consider linear function approximation.
- Define the asymptotic mean-square error to be

$$\text{AMSE}(w) = \lim_{k \to \infty} k\mathbb{E}\left[\|w_k - w^*\|^2\right]$$

Main Results

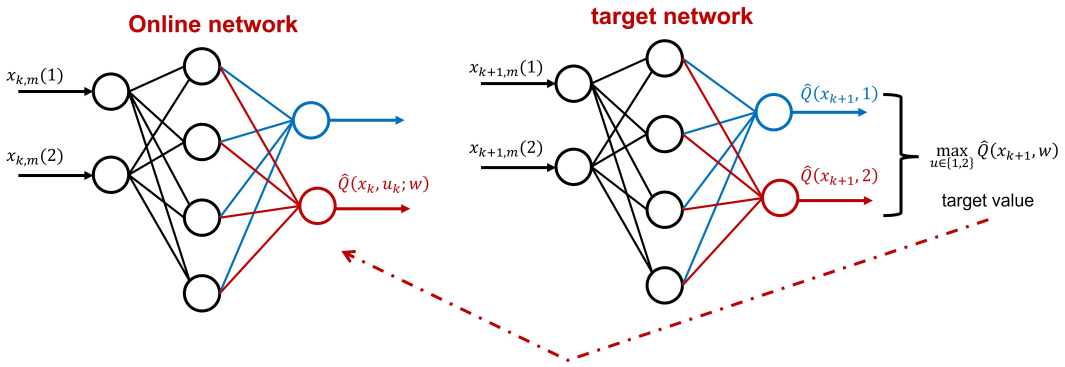
- $$\text{AMSE}\left(w^{(1)}\right) \geq \text{AMSE}\left(w\right).$$
- $$\text{AMSE}\left(\frac{w^{(1)} + w^{(2)}}{2}\right) = \text{AMSE}\left(w\right).$$

Recall the Convergence of Q-Learning

$$E\left[\|Q_{k+1} - Q^*\|_\infty^2\right] \leq (1 - \beta + \alpha\beta))^{2k+2} E\left[\|Q_0 - Q^*\|_\infty\right] + \frac{c\beta^2 \log^2 N}{(1-\rho)N}$$

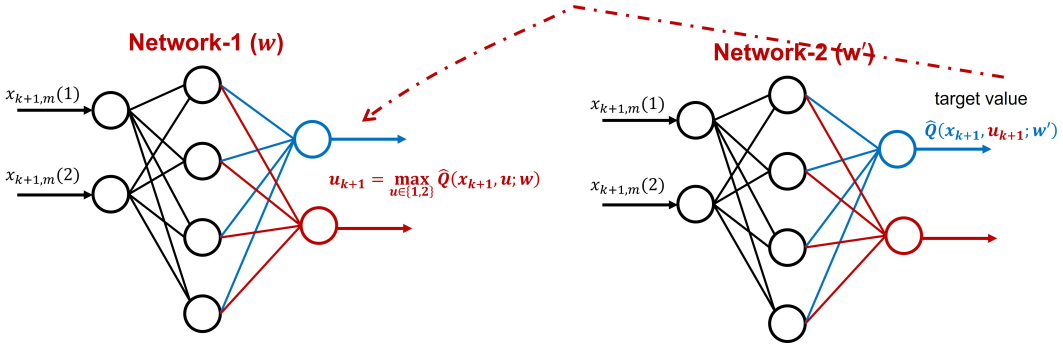
A larger learning rate leads to faster convergence. So Double Q-Learning learns faster without sacrificing AMSE.

Deep-Q-Network (DQN) (Minh et al 2015) in PyTorch



Target network is used to (i) select the action u (ii) evaluate $Q(x_{k+1}, u)$. Selecting the optimal action and evaluating the Q-function together lead to an overestimation.

Double DQN (van Hasselt, Guez, Silver 2015)



Double DQN

Separate action selection and evaluation. The update of the target network remains the same. For $y_{k,b}$, the action is selected by the online network and the value is evaluated by the target network.

Other Ideas: Clipped Double-Q Learning (Fujimoto, van Hoof, David Meger 2018)

Maintain two Q-networks, denoted by their parameters w and w',

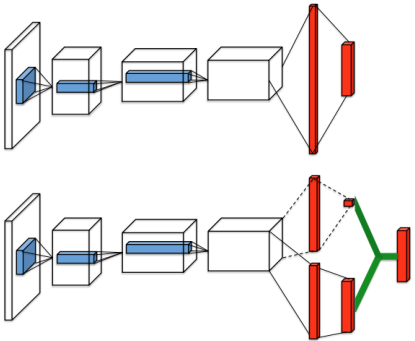
$$w \leftarrow w + \beta_k \sum_b (y_{k,b} - Q(x_{k,b}, u_{k,b}; w)) \nabla_w Q(x_{k,b}, u_{k,b}; w)$$

$$w' \leftarrow w' + \beta_k \sum_b (y_{k,b} - Q(x_{k,b}, u_{k,b}; w')) \nabla_{w'} Q(x_{k,b}, u_{k,b}; w')$$

where

$$y_{k,b} = r_{k,b} + \alpha \min \left\{ \max_u Q(x_{k+1,b}, u; w), \max_u Q(x_{k+1,b}, u; w') \right\}$$

Other Ideas: Dueling Network (Wang, et al. 2016)



Dueling DQN (DDQN)

Advantage function $Q(s, a) = V(s) + A(s, a)$

$$Q(x, u; w, w_v, w_a) = V(x; w, w_v) + A(x, u; w, w_a)$$

Dueling Q Learning

Motivation: Is every action equally important?

- DQN and other methods estimate Q value in one stream
  - Inefficiency in state value update, as all actions' Q values need to be changed.
- Usually, most of the actions are not important, except the optimal action.
  - To improve state value learning efficiency and ignore useless actions, estimate them separately, in terms of state value V and action advantage A
- **State value versus action contrast.**

$$Q(s,a) = V(s) + A(s,a),$$

with $A(s,a)$ capturing only the contrast among actions at s. When actions are nearly tied, $A(s,a) \approx 0$ for all a and the network focuses on $V(s)$; when a few actions are indeed better, the network adjusts A locally without relearning V.

Unidentifiability Problem

- Directly summing $V(s)$ and $A(s, a)$ is not unique.
  - Without a constraint, V and A are not identifiable since $V(s) + c$ and $A(s, a) - c$ yield the same Q.
- This makes the decomposition unidentifiable.

Solutions

Subtract Max:

$$Q(x, u; w) = V(x; w) + \big(A(x, u; w) - \max_{v} A(x, v; w)\big),$$

1. When using a greedy policy, $Q(x, a^*) = V(x)$
2. Enforce A to be zero at the chosen action.

Unidentifiability Problem

- Directly summing $V(s)$ and $A(s,a)$ is not unique.
  - Without a constraint, V and A are not identifiable since $V(s)+c$ and $A(s,a)-c$ yield the same Q.
- This makes the decomposition unidentifiable.

Solutions

Subtract mean:

$$Q(x,u;w) = V(x;w) + \left(A(x,u;w) - \frac{1}{|\mathcal{A}|} \sum_{v} A(x,v;w) \right).$$

1. Alternative of subtract max
2. Loss the original semantics of V and A, and off target by a constant.
3. But increase the stability of optimization

Other Ideas: Multi-Step Target (Sutton and Barto 1998)

Multi-Step Bellman Equation

$$\begin{aligned} Q(x_k, u_k) =& E\left[r(x_k, u_k) + \alpha r(x_{k+1}, \pi^*(x_{k+1})) + \cdots + \alpha^{n-1} r(x_{k+n-1}, \pi^*(x_{k+n-1}))\right] \\ &+ \alpha^n \max_v Q(x_{k+n}, v). \end{aligned}$$

SARSA with Multi-Step Target

Given a mult-step experience $(x_k, u_k, r_k, x_{k+1}, u_{k+1}, r_{k+1}, \cdots, x_{k+n}, u_{k+n}, r_{k+n})$, minimize the following TD error

$$\left(r_k^{(n)} + \alpha^n \max_v Q(x_{k+n}, v; w) - Q(x_k, u_k; w)\right)^2$$

where $r_k^{(n)} = r_k + \alpha r_{k+1} + \cdots + \alpha^{n-1} r_{k+n-1}$.

Other Ideas: Prioritized Replay (Schaul et al. 2016)

Sample an experience based on its absolute TD error

$$\delta_k^\omega = \left| r(x_k, u_k) + \alpha \max_v Q(x_{k+1}, v) - Q(x_k, u_k) \right|^\omega$$

where $\omega > 0$ is a hyper parameter.

Motivation

Prioritize those experiences based on the amount the RL agent can learn from. TD error measures how "surprising" an experience is.

Other Ideas: Prioritized Replay (Schaul et al. 2016)

Implementation (Schaul et al. 2016)

- When a new experience is generated, store it with the highest priority $z_{\max}$.
- Sample m experiences with distribution $p_k = \frac{z_k}{\sum_i z_i}$.
- If experience k is sampled, update its weight $z_k = |\delta_k|^\omega$ (the TD error is computed using the current network parameters. The weight change from experience k is

$$s_j \delta_k \nabla \hat{Q}(x_k, u_k, w),$$

where $s_j = \frac{1}{(Np_k)^\beta \max_i s_i}$, where β is a hyper parameter and s_j is the importance sampling weight to correct the bias introduced in prioritized replay.

Other Ideas: Distributional RL (Bellemare etl al. 2017)

Distributional RL

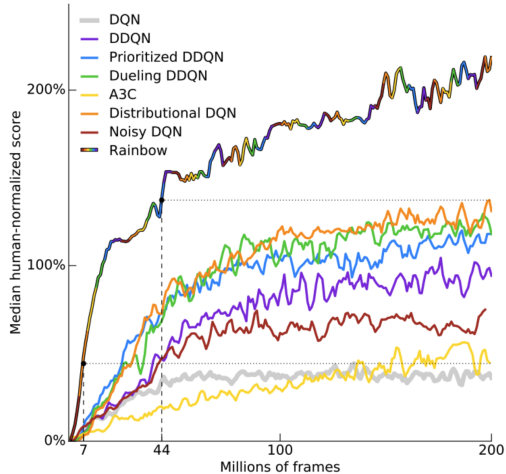
We view the reward function as a random vector $R \in Z$, and define the transition operator $P^\pi : Z \to Z$

$$P^\pi Z(x,u) := Z(X',U'), X' \sim P(\cdot|x,u), U' \sim \pi(\cdot|X').$$

We define the distributional Bellman operator $T^\pi : Z \to Z$ as

$$T^\pi Z(x,u) := R(x,u) + \gamma P^\pi Z(x,u)$$

Rainbow


DQN
DDQN
Prioritized DDQN
Dueling DDQN
A3C
Distributional DQN
Noisy DQN
Rainbow
200%
100%
0%
Median human-normalized score
7
44
100
200
Millions of frames

References

- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. nature, 518(7540), pp.529-533.
- Sebastian Thrun and A. Schwartz, *Issues in Using Function Approximation for Reinforcement Learning*, Proceedings of the Connectionist Models Summer School, June, 1993.
- H Hasselt, *Double Q-learning*, NeurIPS, 2010.
- Wang et al, *Dueling Network Architectures for Deep Reinforcement Learning*, ICML, 2016.
- H. van Hasselt. , A. Guez, D. Silver, *Deep Reinforcement Learning with Double Q-Learning*, AAAI. 2016.
- Scott Fujimoto, Herke van Hoof, and David Meger, *Addressing Function Approximation Error in Actor-Critic Methods*, ICML, 2018.

References

- Weng, W., Gupta, H., He, N., Ying, L. and Srikant, R., 2020. *The mean-squared error of double Q-learning*, NeurIPS, 33, pp.6815-6826.
- Tom Schaul, John Quan, Ioannis Antonoglou and David Silver. Prioritized Experience Replay. ICLR 2016.
- Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. AAAI 2018.
- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2018.