Reinforcement Learning

- Sarath Chandar

8. More on function approximation.

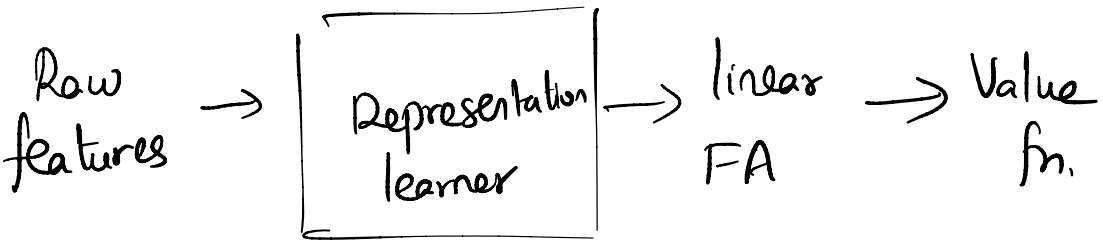
8. More on Function Approximation

Non-linear fn. approximation:-

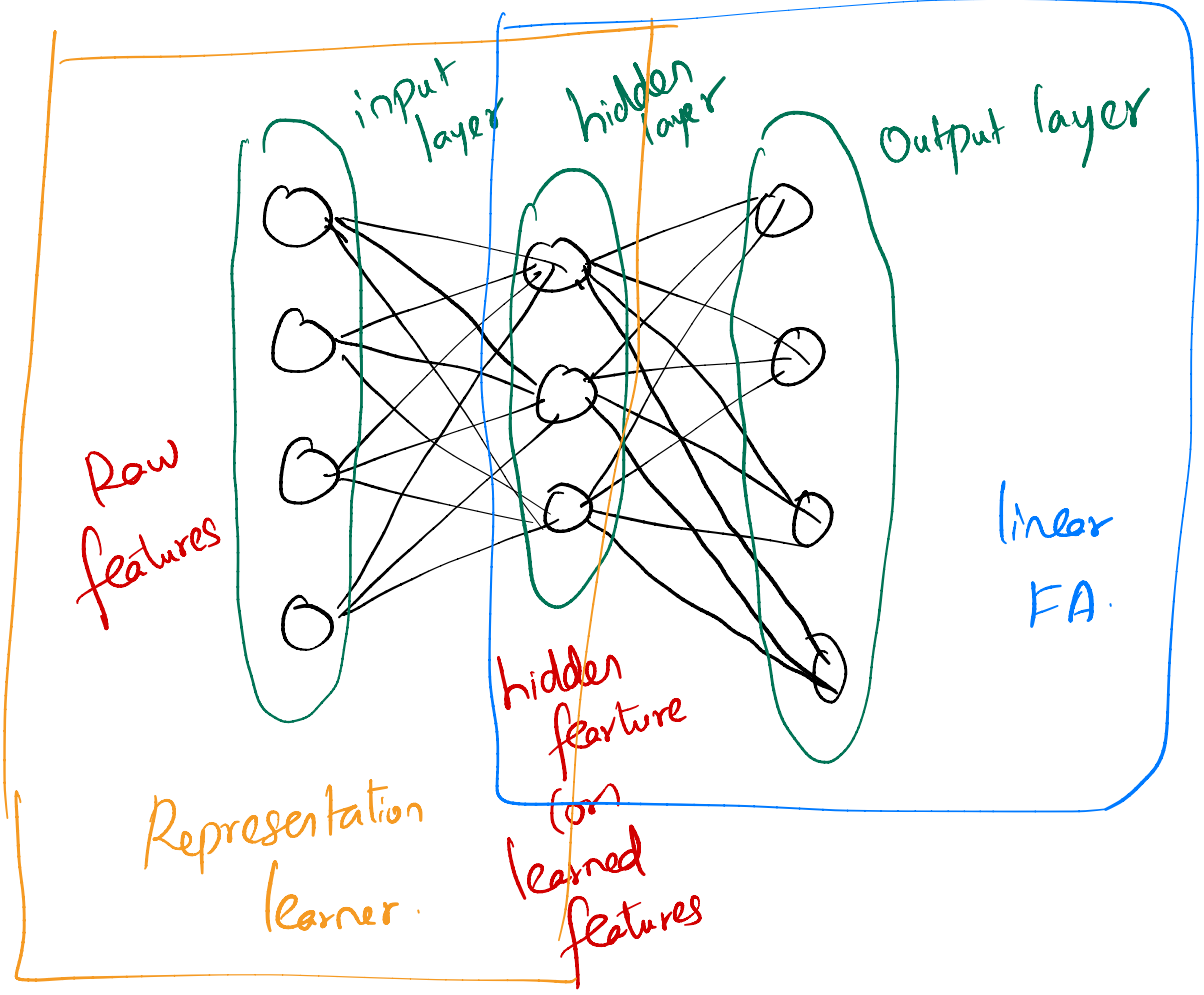
Linear fn approximators we have seen so for requires carefully handcrafted features that makes the learning problem easier. Even then interactions between these features is limited.

Can we automatically learn better features that can be passed on to the linear FA?

Idea of representation learning:-



Neural networks are the most successful class of representation learners.



Each layer has some non-linear activation fns.

→ Non-linear activation fn.

ex:

$$h_1 = \sigma(W_1 x + b_1)$$

$$h_2 = \sigma(W_2 x + b_2)$$

$$O = W_3 x + b_3$$

$W_1, b_1, W_2, b_2, W_3, b_2$

Parameters of the NN.

Non-linear activation fn:

1. Sigmoid $\sigma(a) = \frac{1}{1+e^{-a}}$
2. tanh $\tanh(a) = (e^a - e^{-a})/(e^a + e^{-a})$
3. ReLU $\text{ReLU}(a) = \max(0, a)$

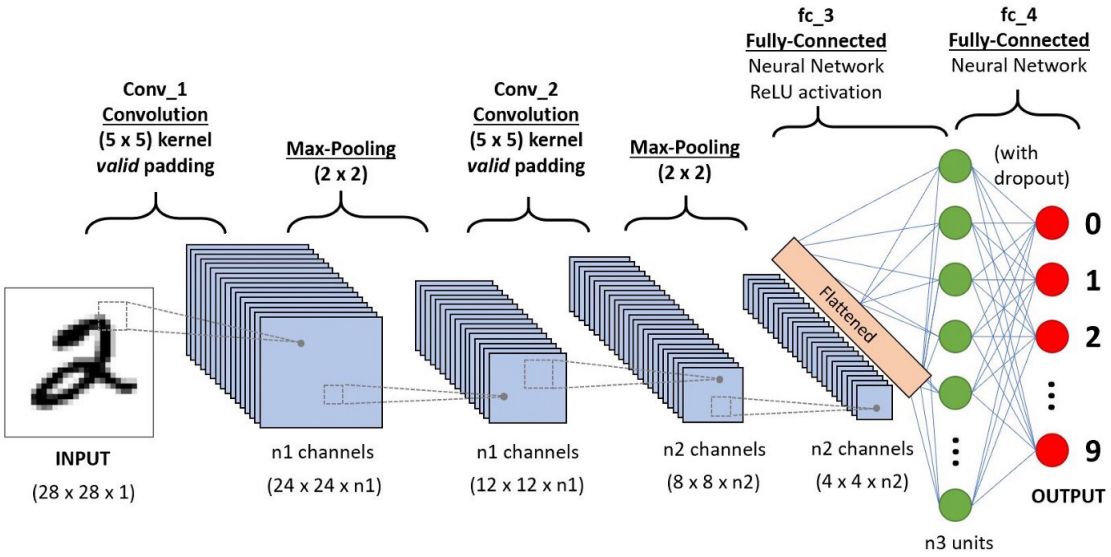
Neural networks can be trained using gradient descent. The most efficient way to compute gradients for weights in each layer is the idea of backpropagation. Backpropagation exploits the chain rule.

$$\frac{\partial \text{loss}}{\partial W_2} = \frac{\partial \text{loss}}{\partial o} \frac{\partial o}{\partial h_2} \frac{\partial h_2}{\partial W_2}$$

This is an example for fully connected Multi-layer Perceptron (MLP) network.

Convolutional Neural Networks:- (CNNs)

When we have image observations, we can use (CNNs) that exploit the spatial structure of the pixels in the image.


fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
(with
dropout)
Flattened
0
1
2
9
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
OUTPUT

Off-Policy methods with function approximation.

The extension to function approximation turns out to be significantly different and harder for off-Policy learning than it is for off-Policy learning. Tabular off-policy methods while easy to extend to gradient based methods, do not converge as robustly as they do under on-policy training.

Quick recap of off-Policy learning :- The goal of off-Policy learning is to learn a value fn for a target policy π, given data due to a different behavior policy b.

In prediction → π, b are static and given.

In control → we learn action values and both π, b change during learning.

π - greedy policy w.r.t. $\hat{q}$

b - more exploratory like ϵ-greedy w.r.t. $\hat{q}$.

Two challenges in Off-Policy learning:-

① target of the update (arises in both tabular and FA settings)

② distribution of updates (arises only with FA).

Techniques related to importance sampling deal with the first challenge.

Challenge #2: The distribution of updates in the off-policy case is not according to the on-policy distribution. The on-policy distribution is important to the stability of the semi-gradient methods.

Semi-gradient methods:-

We will first extend the off-policy methods developed before to FA as semi-gradient methods. Note that these methods only address the first challenge and hence might diverge.

To convert tabular off-Policy algorithms to Semi-gradient form, we simply replace the update to an array (V or Q) to an update to a weight vector (w), using the approximate value fn ($\hat{v}$ or $\hat{q}$) and its gradient.

Per-step importance Sampling ratio :

$$\rho_t = \rho_{t:t} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Semi-gradient off-Policy TD :-

- one-step, state-value algorithm

$$w_{t+1} = w_t + \alpha \rho_t \delta_t \nabla \hat{v}(S_t, w_t)$$

where $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t)$

Semi-gradient Expected Sarsa :-

- one-step, action-value algorithm.

$$w_{t+1} = w_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t; w_t)$$

where $\delta_t = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a; w_t) - \hat{q}(S_t, A_t; w_t)$.

Note that this algorithm does not use importance sampling. This is clear in tabular case since only sample action is A_t and in learning its value we do not have to consider any other actions.

Multi-step generalization of both state-value and action-value algorithms involve importance sampling.

n-step semi-gradient expected Sarsa:-

$$w_{t+n} = w_{t+n-1} + \alpha \rho_{t+1} \rho_{t+2} \cdots \rho_{t+n-1} \left[G_{t:t+n} - \hat{q}(S_t, A_t; w_{t+n-1}) \right] \nabla \hat{q}(S_t, A_t; w_{t+n-1}).$$

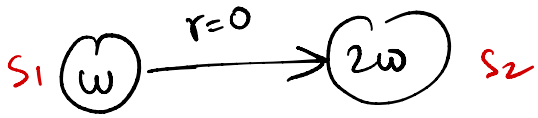
with

$$G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}; w_{t+n-1})$$

Examples of Off-Policy divergence :-

Second challenge - distribution of updates does not match the on-policy distribution.

Example 1: As a part of larger MDP, there are 2 states whose estimate values are of functional form w and $2w$.



Suppose initally $w = 10$.

$\hat{v}(S_1; w) = 10 \quad \hat{v}(S_2; w) = 20$.

'w' will be increased to raise the first state's value.

if $\gamma \sim 1$, TD error ~ 10 and if $\alpha = 0.1$, w will be increased to nearly 11.

Now, $\hat{v}(S_1; w) = 11 \quad \hat{v}(S_2; w) = 22$.

TD error = 11 (larger)

new 'w' ≈ 12.1

These updates will diverge to infinity!

TD error:

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}; w_t) - \hat{v}(S_t; w_t)$$

$$= 0 + \gamma 2w_t - w_t = (2\gamma - 1) w_t .$$

Off-policy semi-gradient TD update:

$$w_{t+1} = w_t + \alpha \rho_t \nabla \delta_t \hat{v}(S_t; w_t)$$

$$= w_t + \alpha \, 1 \, (2\gamma - 1) w_t \cdot 1$$

$$= \left(1 + \alpha(2\gamma - 1)\right) w_t$$

If $\gamma > 0.5$, then $\left(1 + \alpha(2\gamma - 1)\right) > 1$ and hence the system is unstable.

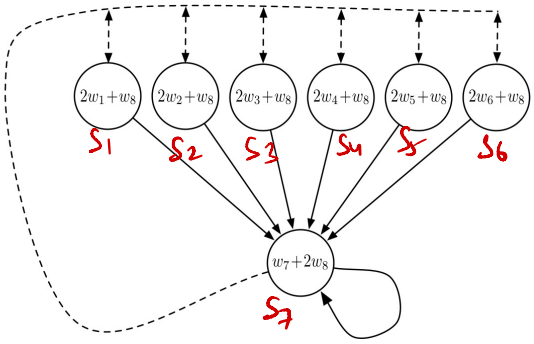
Note: Stability does not depend on α.

This issue happens because one transition occurs repeatedly without w being updated on other transitions. This is possible under off-policy training but not under on-policy training.

In on-policy training, each time there

is a transition from S_1 to S_2, there should also be a transition out of S_2 [since ρ_t is always 1 in on-policy case]. That transition would reduce w.

Example 2: A complete example by Baird.



$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$

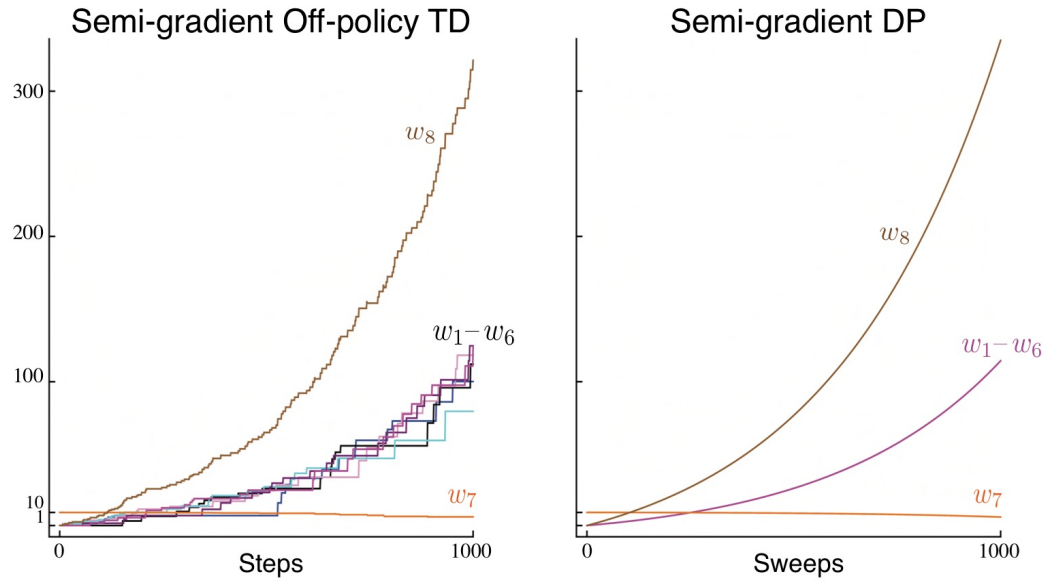
Reward is 0 on all transitions.

$w \in \mathbb{R}^8$ $\quad S_1 : 2w_1 + w_8$

$x(1) = (2, 0, 0, 0, 0, 0, 0, 1)^T$

True $v_\pi(s) = 0$ for all s. $\Rightarrow$ $w = 0$.

Set of feature vectors is linearly independent set.



Semi-gradient DP update:

$$w_{k+1} = w_k + \frac{\alpha}{|S|} \sum_s \left(\mathbb{E}_\pi [R_{t+1} + \gamma \hat{v}(S_{t+1}, w_k) | S_t = s] - \hat{v}(s, w_k) \right) \nabla \hat{v}(s, w_k)$$

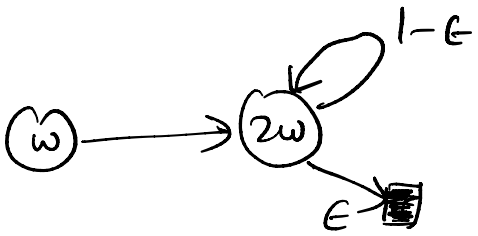
This update does not have randomness and asynchrony. Yet still the system is unstable!

If we alter the distribution of DP updates from uniform to on-policy distribution, then convergence is guaranteed.

Even simplest combination of bootstrapping and FA can be unstable if the updates are not based on on-policy distribution.

Suppose we change the value fn all the way to the best least-squares approx. instead of taking a step toward the expected one-step return. Would that solve the instability problem?

Example 3: Tsitsiklis and Van Roy's counterexample:-



reward is 0 in all transitions.

$$w_{k+1} = \underset{w \in \mathbb{R}}{\operatorname{argmin}} \sum_{s \in S} \left(\hat{v}(s;w) - \mathbb{E}_\pi \left[R_{t+1} + \gamma \hat{v}(S_{t+1}; w_k) \middle| S_t = s \right] \right)^2$$

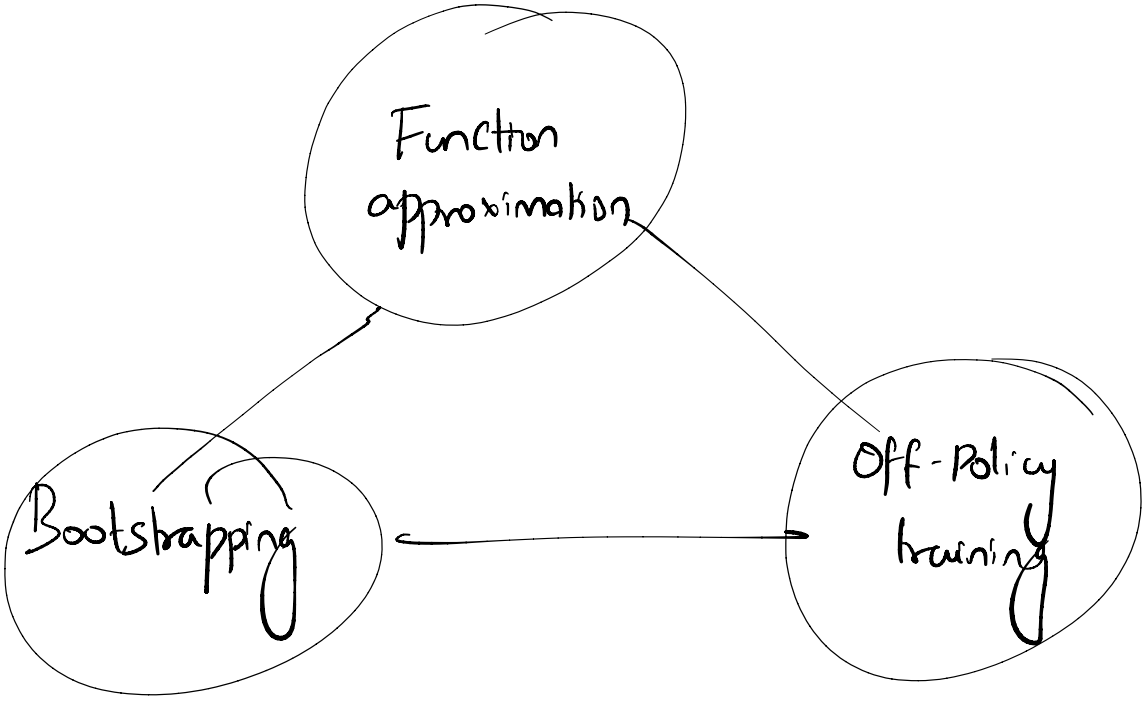
$$= \underset{w \in \mathbb{R}}{\operatorname{argmin}} \left(w - \gamma 2 w_k \right)^2 + \left(2w - (1-\epsilon)\gamma 2 w_k \right)^2$$

$$= \frac{6-4\epsilon}{5} \gamma w_k .$$

The sequence $\{w_k\}$ diverges when $\gamma > \frac{5}{6-4\epsilon}$ and $w_0 \neq 0$.

The deadly triad :-

The danger of instability and divergence arises whenever we combine all the following 3 elements:



Note: The danger is not due to control or to GPT. Instabilities arises even in prediction.

The danger is not due to learning or to uncertainties about the env. They occur in DP too!

Convergence of prediction algorithms :-

on/off-Policy	Algor.	Tabular	linear FA	Non-linear FA
On-policy	MC	✓	✓	✓
	TD	✓	✓	X
Off-policy	MC	✓	✓	✓
	TD	✓	X	X

If any two elements of the deadly triad are present, but not all three, then instability can be avoided.

FA: Cant be given up. We need ability to scale to large problems.

Bootstrapping: Needed for computational efficiency and data efficiency.

Off-policy learning: On-policy learning is often adequate. However off-policy learning is needed

if we want to learn about many tasks from one stream of experience.

Deep Q-Networks :- (DQN)

Deep Q-Networks is a Combination of Q-Learning and Neural network function approximator.

Q-learning with FA can diverge. DQN uses two tricks to avoid this.

① Experience Replay:

DQN stores the agent's experiences at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$ in a buffer pooled over many episodes called "replay buffer". Q-learning updates are computed by sampling a min-batch of transitions from the buffer.

Advantages:

1. Each step of experience is potentially used in many weight updates → data efficiency.
2. Randomizing samples break correlations and reduces the variance of updates.
3. By using the replay buffer, the behavior distribution is averaged over many of its prev. states, avoiding oscillations or divergence.

Note: Buffer size is limited and transitions are removed in First In First Out (FIFO) fashion.

(2) Target Network :-

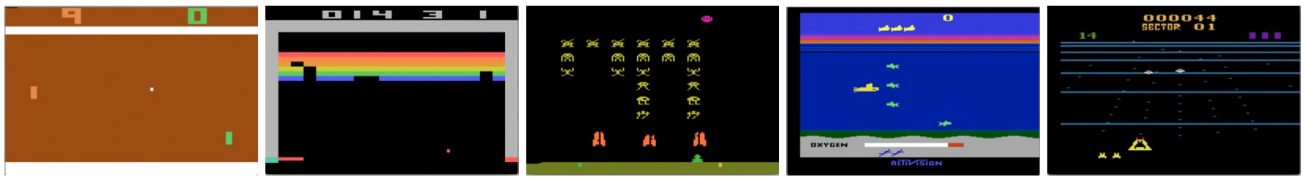
DQN keeps a separate target network θ^- to compute the targets. This is same as the online network θ and its parameters

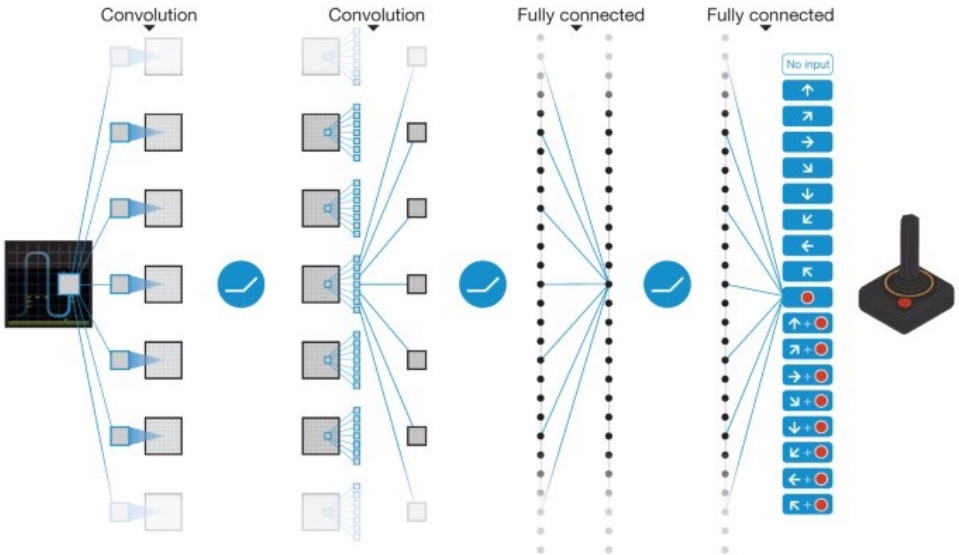
are copied from online network for every τ timesteps.

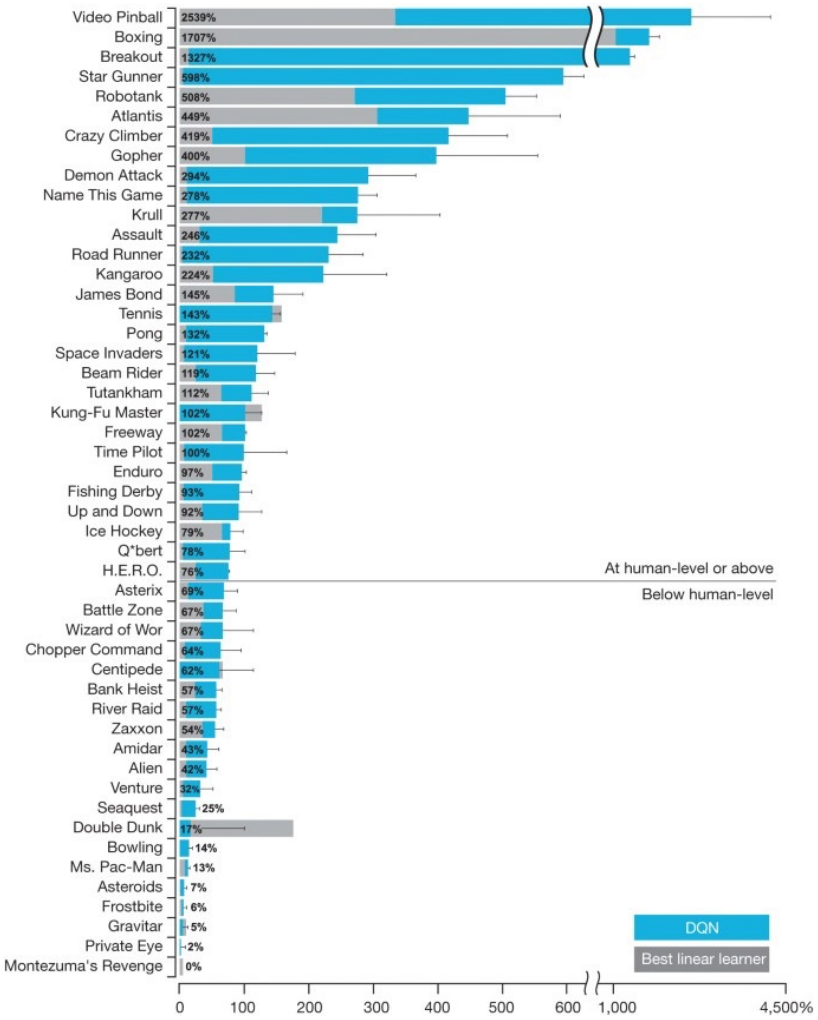
$$\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-) - Q(S_t, A_t; \theta_t)$$

DQN also clips the rewards to $(-1, 1)$ which further helps in reducing divergence.

DQN for Atari :-






Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Average reward setting for Continuing tasks :-

Consider Continuing tasks but without discounting.

The quality of a policy π is defined as the average rate of reward (or average reward) while following that policy.

$$r(\pi) = \lim_{h \to \infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}\left[R_t \mid S_0, A_{0:t-1} \sim \pi \right]$$

$$= \lim_{t \to \infty} \mathbb{E}\left[R_t \mid S_0, A_{0:t-1} \sim \pi \right]$$

$$= \sum_{s} \mu_\pi(s) \sum_{a} \pi(a|s) \sum_{s', r} p(s', r | s, a)\, r.$$

μ_π is the steady-state distribution.

$$\mu_\pi(s) = \lim_{t \to \infty} \Pr\{ S_t = s \mid A_{0:t-1} \sim \pi \}$$

which is assumed to exist for any π and to be independent of S_0. [Ergodicy assumption for MDP.]

It means that where the MDP starts or any early decision made by the agent can have only a temporary effect. Ergodicity is sufficient to guarantee the existence of the limits in the equations above.

Steady state distribution means

$$\sum_s \mu_\pi(s) \sum_a \pi(a|s)\, p(s'|s,a) = \mu_\pi(s')$$

Returns in Average-reward setting:-

$$G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots$$

differential return.

Corresp. value fns are differential value functions.

Bellman equations:-

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r\,s'} p(s', r|s, a)\left[r - r(\pi) + v_\pi(s')\right]$$

$$q_{\pi}(s,a) = \sum_{r,s'} p(s',r \mid s,a)\left[r - r(\pi) + \sum_{a'} \pi(a'|s')\, q_{\pi}(s',a')\right]$$

$$v_*(s) = \max_a \sum_{r,s'} p(s',r \mid s,a)\left[r - \max_{\pi} r(\pi) + v_*(s')\right]$$

$$q_*(s,a) = \sum_{r,s'} p(s',r \mid s,a)\left[r - \max_{\pi} r(\pi) + \max_{a'} q_*(s',a')\right]$$

$$\delta_t = R_{t+1} - \bar{R}_{t+1} + \hat{v}(S_{t+1}; w_t) - \hat{v}(S_t; w_t)$$

$$\delta_t = R_{t+1} - \bar{R}_{t+1} + \hat{q}(S_{t+1}, A_{t+1}; w_t) - \hat{q}(S_t, A_t; w_t)$$

where $\bar{R}_t$ is an estimate of average reward at time t.

Example:-

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step sizes $\alpha, \beta > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state S, and action A
Loop for each step:
  Take action A, observe R, S'
  Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
  $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \beta\delta$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
  $S \leftarrow S'$
  $A \leftarrow A'$

Depreciating the discounted setting :-

The continuing, discounted problem formulation has been very useful in the tabular case. But it is questionable in the approximate case.

Consider an infinite sequence of returns with no beginning or end and no clearly identified states. Consider all feature vectors are same. Thus one has only the reward sequence and performance has to be purely assessed from these.

One way → average the reward over a long interval. [idea of average reward setting]

Another way → idea of discounting can be used. Some returns will be small and some will be large. So again we have to average them.

If turns out that in this setting, average of the discounted returns is proportional to the average reward.

For policy π, average of the discounted returns is always $r(\pi)/(1-\gamma)$. Ordering of all policies is not affected by γ. γ has no effect in problem formulation.

discounted value fn.

$$J(\pi) = \sum_s \mu_\pi(s)\, v_\pi^\gamma(s)$$

$$= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi^\gamma(s')\right]$$

$$= r(\pi) + \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\,\gamma v_\pi^\gamma(s')$$

$$= r(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') \sum_s \mu_\pi(s) \sum_a \pi(a|s)\, p(s'|s,a)$$

$$= r(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s')\, \mu_\pi(s')$$

$$= r(\pi) + \gamma J(\pi)$$

$$= r(\pi) + \gamma\, r(\pi) + \gamma^2 J(\pi)$$

$$= r(\pi) + \gamma\, r(\pi) + \gamma^2 r(\pi) + \ldots$$

$$= \frac{1}{1-\gamma}\, r(\pi).$$

Symmetry argument: Each time step is exactly same as every other.

weight of t^{th} reward : $1 + \gamma + \gamma^2 + \ldots + \ldots$

$$= \frac{1}{1-\gamma}.$$

All states are same.

$$\text{average return} = \frac{r(\pi)}{(1-\gamma)}$$

Discounting has no role in definition of the Control problem with FA. One can nevertheless use γ in FA. Then γ changes from problem parameter to a soln method parameter!

The root cause of the difficulties with the discounted control setting is that with FA, we lost the policy improvement theorem!