Dueling Network Architectures for Deep Reinforcement Learning (ICML 2016)

Yoonho Lee

Department of Computer Science and Engineering
Pohang University of Science and Technology

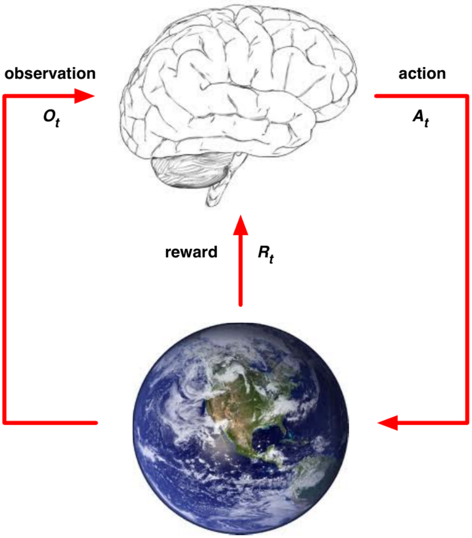
October 11, 2016

Outline

Outline

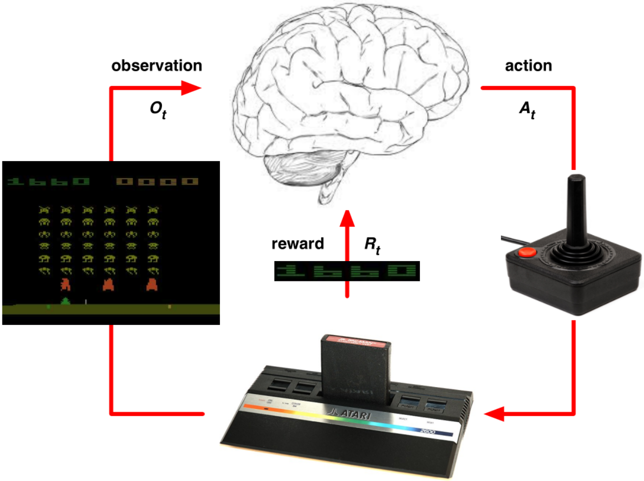
Definition of RL

general setting of RL



Definition of RL

atari setting



POSTECH

Outline

Reinforcement Learning

Definition of RL

Mathematical formulations

Algorithms

Neural Fitted Q Iteration

Deep Q Network

Double Deep Q Network

Prioritized Experience Replay

Dueling Network

Mathematical formulations

objective of RL

Definition

Return G_t is the cumulative discounted reward from time t

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$

Mathematical formulations

objective of RL

Definition

Return G_t is the cumulative discounted reward from time t

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$$

Definition

A policy π is a function that selects actions given states

$$\pi(s) = a$$

- The goal of RL is to find π that maximizes G_0

Mathematical formulations

Q-Value

$$G_t = \sum_{i=0}^{\infty} \gamma^k r_{t+i+1}$$

Definition

The action-value (Q-value) function $Q_\pi(s, a)$ is the expectation of G_t under taking action a, and then following policy π afterwards

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a, A_{t+i} = \pi(S_{t+i}) \forall i \in \mathbb{N}]$$

- "How good is action a in state s?"

Mathematical formulations

Optimal Q-value

Definition

The optimal Q-value function $Q_*(s, a)$ is the maximum Q-value over all policies

$$Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

Mathematical formulations

Optimal Q-value

Definition

The optimal Q-value function $Q_*(s, a)$ is the maximum Q-value over all policies

$$Q_*(s, a) = \max_{\pi} Q_\pi(s, a)$$

Theorem

There exists a policy π_* such that $Q_{\pi_*}(s, a) = Q_*(s, a)$ for all s, a

- Thus, it suffices to find Q_*

Mathematical formulations

Q-Learning

Bellman Optimality Equation

$Q_*(s, a)$ satisfies the following equation:

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q_*(s', a')$$

Bellman Optimality Equation

$Q_*(s, a)$ satisfies the following equation:

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q_*(s', a')$$

Q-Learning

Let a be ϵ-greedy w.r.t. Q, and a' be optimal w.r.t. Q. Q converges to Q_* if we iteratively apply the following update:

$$Q(s, a) \leftarrow \alpha(R(s, a) + \gamma Q(s', a')) + (1 - \alpha)Q(s, a)$$

Mathematical formulations

Other approcahes to RL

Value-Based RL

- Estimate $Q_*(s, a)$
- Deep Q Network

Policy-Based RL

- Search directly for optimal policy π_*
- DDPG, TRPO...

Model-Based RL

- Use the (learned or given) transition model of environment
- Tree Search, DYNA ...

Outline

Neural Fitted Q Iteration

```
NFQ_main() {
input: a set of transition samples D; output: Q-value function Q_N
    k=0
    init_MLP() → Q_0;
    Do {
        generate_pattern_set P = {(input^l, target^l), l = 1, ..., #D} where:
            input^l = s^l, u^l,
            target^l = c(s^l, u^l, s'^l) + γ min_b Q_k(s'^l, b)
        Rprop_training(P) → Q_{k+1}
        k:= k+1
    } While (k < N)
```

Fig. 1. Main loop of NFQ

- Supervised learning on (s,a,r,s')

Neural Fitted Q Iteration

input

NFQ_main() {
$\boxed{\text{input: a set of transition samples } D\text{; output: Q-value function } Q_N}$

k=0

init_MLP() $\rightarrow Q_0$;

Do {

generate_pattern_set $P = \{(input^l, target^l), l = 1, \ldots, \#D\}$ where:

$input^l = s^l, u^l$,

$target^l = c(s^l, u^l, s'^l) + \gamma\, min_b Q_k(s'^l, b)$

Rprop_training$(P) \rightarrow Q_{k+1}$

k:= k+1

} While $(k < N)$

Fig. 1. Main loop of NFQ

Neural Fitted Q Iteration

target equation

NFQ_main() {
input: a set of transition samples D; output: Q-value function Q_N

k=0

init_MLP() $\rightarrow Q_0$;

Do {

generate_pattern_set $P = \{(input^l, target^l), l = 1, \ldots, \#D\}$ where:

$$\boxed{\begin{aligned} &input^l = s^l, u^l, \\ &target^l = c(s^l, u^l, s'^l) + \gamma\, min_b Q_k(s'^l, b) \end{aligned}}$$

Rprop_training(P) $\rightarrow Q_{k+1}$

k:= k+1

} While $(k < N)$

Fig. 1. Main loop of NFQ

Neural Fitted Q Iteration

Shortcomings

- Exploration is independent of experience
- Exploitation does not occur at all
- Policy evaluation does not occur at all
- Even in exact(table lookup) case, not guaranteed to converge to Q_*

Outline

Deep Q Network

action policy

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    $\boxed{\text{With probability } \varepsilon \text{ select a random action } a_t}$
    $\boxed{\text{otherwise select } a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)}$
    $\boxed{\text{Execute action } a_t \text{ in emulator and observe reward } r_t \text{ and image } x_{t+1}}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

- Choose an ϵ-greedy policy w.r.t. Q

Deep Q Network

network freezing

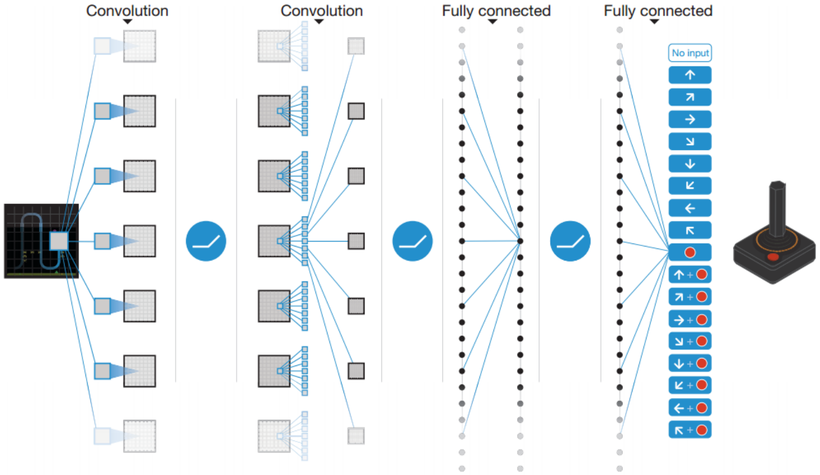
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

- Get a copy of the network every C steps for stability

Deep Q Network

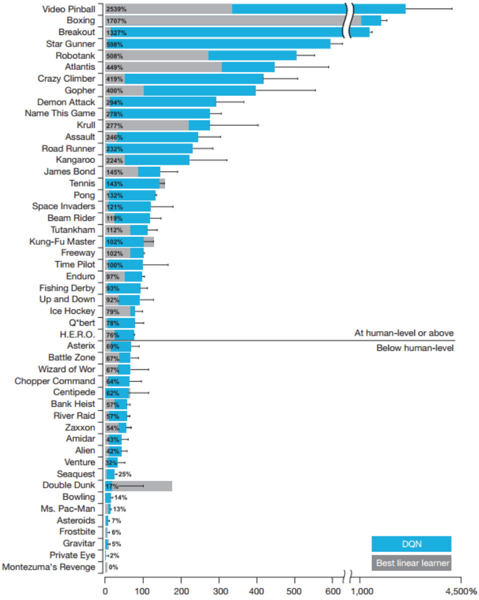
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Deep Q Network



Deep Q Network

performance



POSTECH

Deep Q Network

overoptimism

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \boxed{\hat{Q}\left(\phi_{j+1}, a'; \theta^-\right)} & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

- What happens when we overestimate Q?

Deep Q Network

Problems

- Overestimation of the Q function at any s spills over to actions that lead to $s \rightarrow$ Double DQN
- Sampling transitions uniformly from D is inefficient $\rightarrow$ Prioritized Experience Replay

Outline

Double Deep Q Network

target

We can write the DQN target as:

$$Y_t^{DQN} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{argmax} Q(S_{t+1}, a; \theta_t^-); \theta_t^-)$$

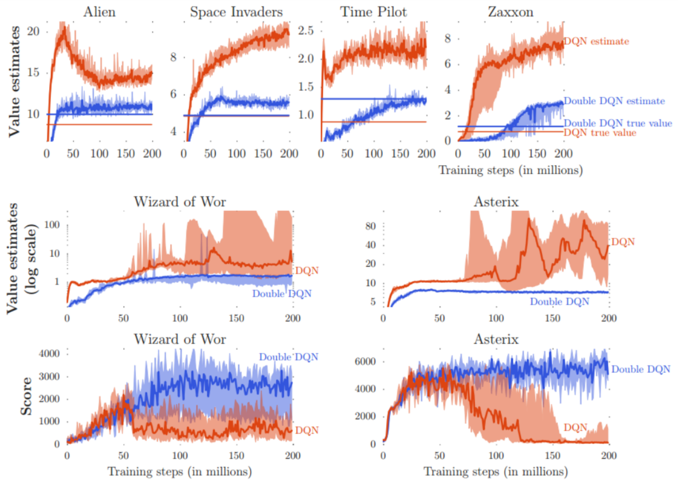
Double DQN's target is:

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{argmax} Q(S_{t+1}, a; \theta_t); \theta_t^-)$$

This has the effect of decoupling action selection and action evaluation

Double Deep Q Network

performance



- Much stabler with very little change in code

Outline

Prioritized Experience Replay

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

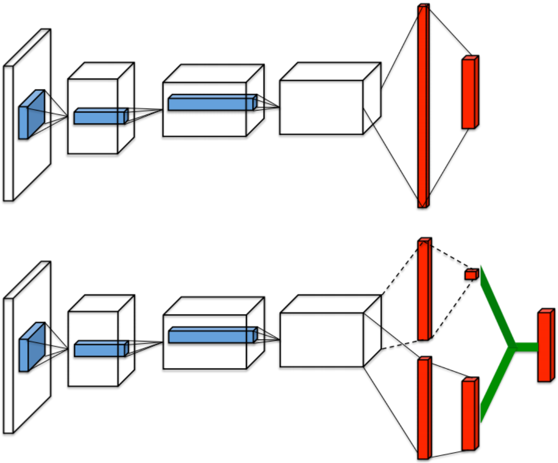
- Update 'more surprising' experiences more frequently

Outline

Reinforcement Learning
- Definition of RL
- Mathematical formulations

Algorithms
- Neural Fitted Q Iteration
- Deep Q Network
- Double Deep Q Network
- Prioritized Experience Replay
- Dueling Network

Dueling Network



- The scalar approximates V, and the vector approximates A

Dueling Network

forward pass equation

The exact forward pass equation is:

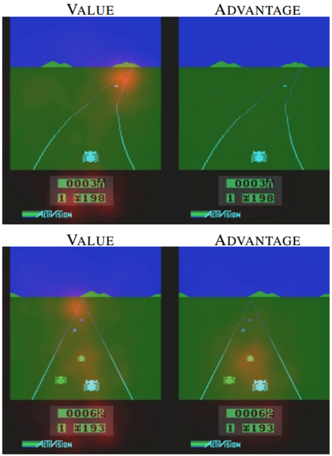
$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \max_{a'} A(s, a'; \theta, \alpha))$$

The following module was found to be more stable without losing much performance:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha))$$

Dueling Network

attention



- Advantage network only pays attention when current action crucial

Dueling Network

performance

	30 no-ops		Human Starts	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%

- Achieves state of the art in the Atari domain among DQN algorithms

Dueling Network

Summary

- Since this is an improvement only in network architecture, methods that improve DQN(e.g. Double DQN) are all applicable here as well
- Solves problem of V and A typically being of different scale
- Updates Q values more frequently than a single-stream DQN, where only a single Q value is updated for each observation
- Implicitly splits the credit assignment problem into a recursive binary problem of "now or later"

Dueling Network

Shortcomings

- Only works for $|A| < \infty$
- Still not able to solve tasks involving long-term planning
- Better than DQN, but sample complexity is still high
- ϵ-greedy exploration is essentially random guessing