Deep Reinforcement Learning Networks

I-Chen Wu

- Acknowledgement:
  the slides are done by 蔡承倫.

Outline

- Reference
- DQN
- DDQN (Double DQN)
- Actor-Critic (Discrete actions)
- Actor-Critic (Continuous actions)
- Dueling Network
- A3C (Asynchronous Advantage Actor-Critic)

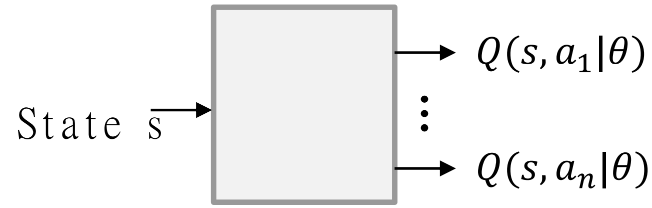
Reference

- DQN:
  - Playing Atari with Deep Reinforcement Learning
    - *Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller*
    - DeepMind Technologies
- Double DQN:
  - Deep Reinforcement Learning with Double Q-learning
    - Hado van Hasselt, Arthur Guez, David Silver
    - Google DeepMind
- Actor-Critic (Discrete actions):
  - Actor-Critic Algorithms
    - Vijay R. Konda John N. Tsitsiklis
    - Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 02139.

Reference

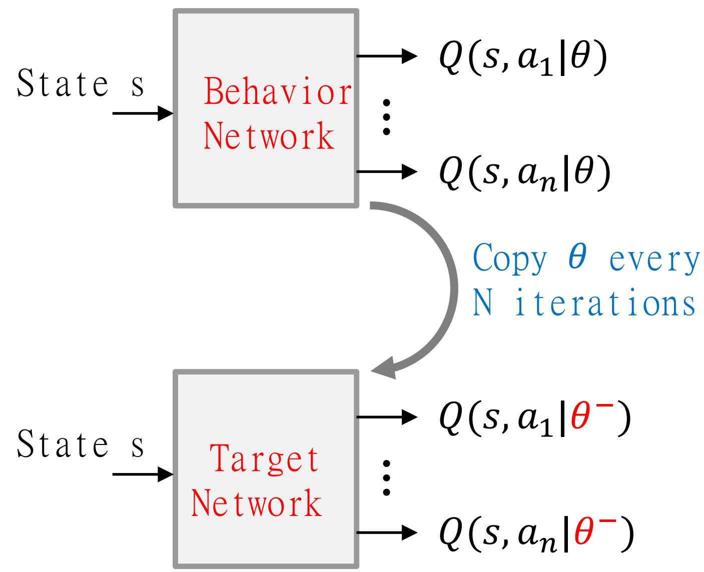
- Actor-Critic (Continuous actions):
  - Continuous control with deep reinforcement learning
    - Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra
    - Google Deepmind London, UK
- Dueling Network :
  - Dueling Network Architectures for Deep Reinforcement Learning
    - *Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas*
    - Google DeepMind London, UK
- A3C :
  - Asynchronous Methods for Deep Reinforcement Learning
    - *Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu*
    - Google DeepMind, Montreal Institute for Learning Algorithms (MILA), University of Montreal

Deep Q Network (DQN)



- Single deep network estimates the action value function of each discrete action
  - Action Value: $Q(s_t, a_t|\theta)$
  - Select action: $\arg\max_{a'} Q(s_t, a'|\theta)$
- Target Q (A real number):
  - $Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'|\theta)$
- Loss Function:
  - $L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right)^2$
- Gradient descent:
  - $\nabla_\theta L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right) \nabla_\theta Q(s_t, a_t|\theta)$

Deep Q Network (DQN)



- Common techniques
  1. Target Network with parameters: θ^-
  2. Experience Replay
  3. ε-greedy

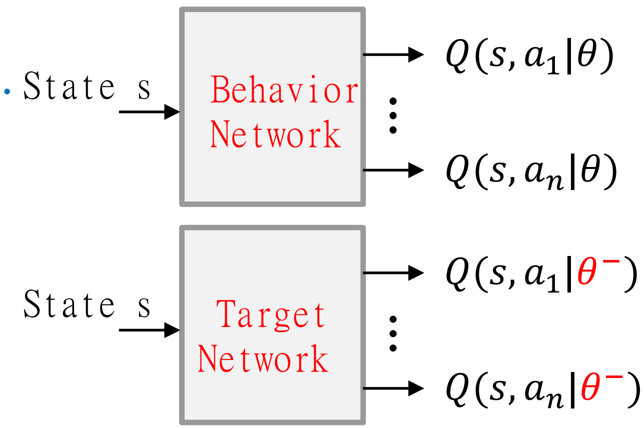
- Apply Target Network on DQN:
  - $Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'|\theta^-)$
  - Gradient descent on behavior network:
    - $\nabla_\theta L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right) \nabla_\theta Q(s_y, a_t|\theta)$
  - Copy parameters from θ to θ^- every N iterations(updates). (Ex. N=1000)

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

DDQN (Double DQN)

- Prevent overoptimistic value estimates on DQN.
- Decouple the selection from the evaluation.



$$Y_t^{Q} = r_{t+1} + \gamma \max_a Q(S_{t+1}, \boxed{a} | \theta^-)$$

$$Y_t^{DoubleQ} = r_{t+1} + \gamma Q\left(S_{t+1}, \boxed{\underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a|\theta)} | \theta^-\right)$$

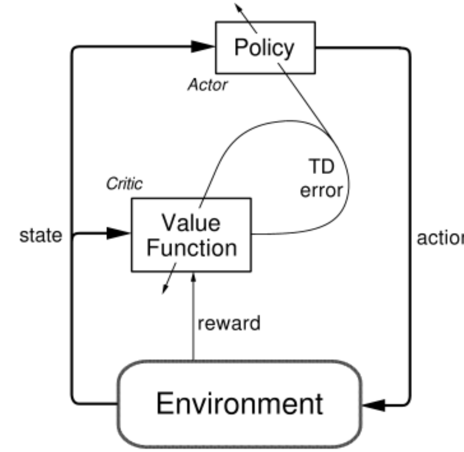
Algorithm 1: Double DQN Algorithm.

input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, \ldots\}$ **do**
    Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
    Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
      replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$

$$y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$$

    Do a gradient descent step with loss $\|y_j - Q(s, a; \theta)\|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end

Actor-Critic (Discrete Actions)



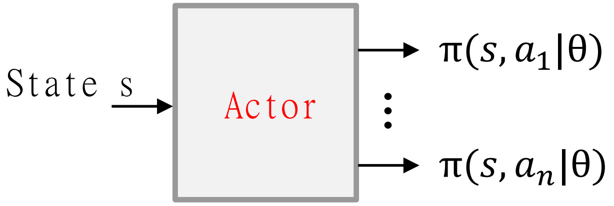
- Use two networks: an actor and a critic
  - Critic estimates value of current policy by Q-learni
    - Gradient:

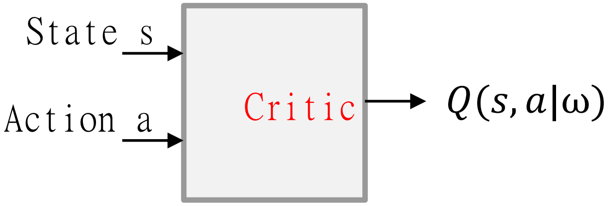
$$\nabla_\omega L_Q(s_t,a_t|\omega) = \left(\left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a'|\omega)\right) - Q(s_t,a_t|\omega)\right)\nabla_\omega Q(s_t,a_t|\omega)$$

  - Actor updates policy in direction that improves Q
    - Gradient (approximate policy gradient):

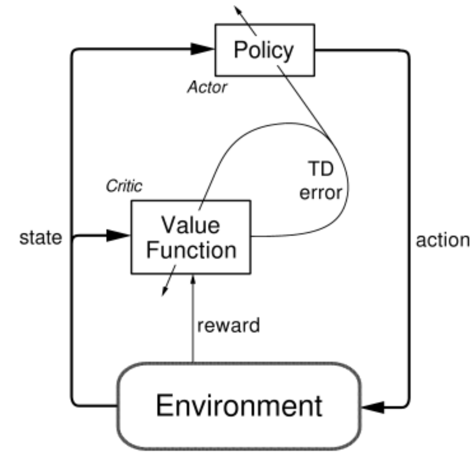
$$J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[Q(s,a|\omega)]$$

$$\nabla_\theta J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[\nabla_\theta \log \pi(s_t,a_t|\theta)\, Q(s_t,a_t|\omega)]$$





Actor-Critic (Continuous Actions)

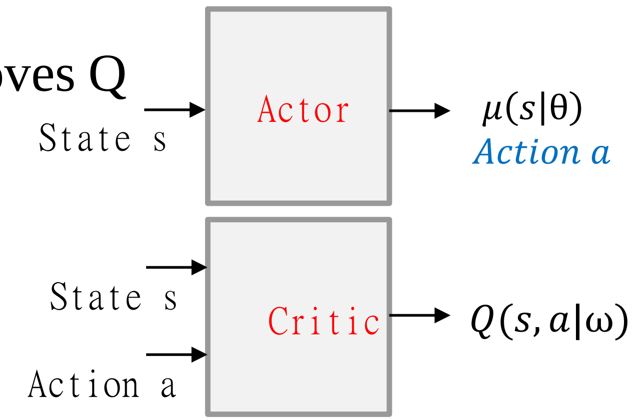


- Use two networks: an actor and a critic
  - Critic estimates value of current action by Q-learni
    - Gradient:

$$\nabla_\omega L_Q(s_t, a_t|\omega) = \left(\left(r_{t+1} + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta)|\omega)\right) - Q(s_t, a_t|\omega)\right) \nabla_\omega Q(s_t, a_t|\omega)$$

  - Actor updates policy in direction that improves Q
    - Gradient (DDPG):

$$\nabla_\theta \mu \approx \mathbb{E}_\mu[\nabla_\theta Q(s_t, \mu(s_t|\theta)|\omega)] = \mathbb{E}_\mu\left[\nabla_a Q(s_t, a|\omega)\Big|_{a=\mu(s_t|\theta)} \nabla_\theta \mu(s_t|\theta)\right]$$



Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

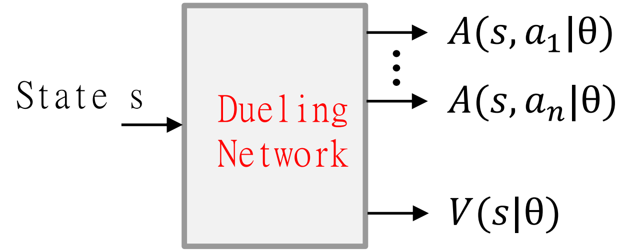
    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

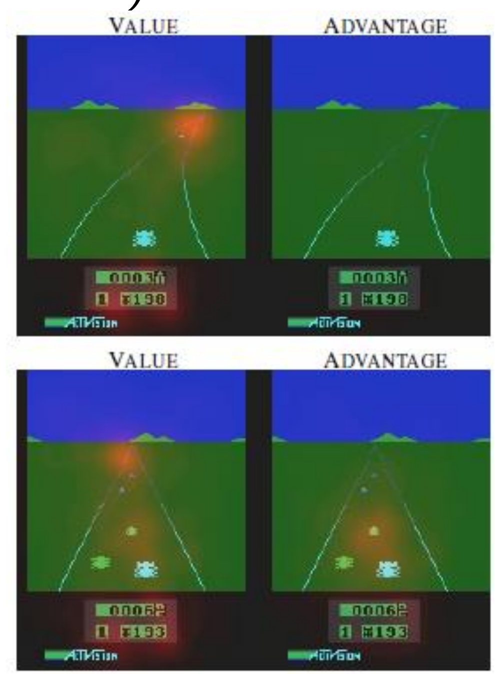
Dueling Network



- A relative measure of the importance of each action
  - $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$
- Improvement (Increase stability)
  - $Q(s_t, a_t|\theta) = V(s_t|\theta) + \left(A(s_t, a_t|\theta) - \frac{1}{|A|}\sum_{a' \in A} A(s_t, a'|\theta)\right)$
- Update network:

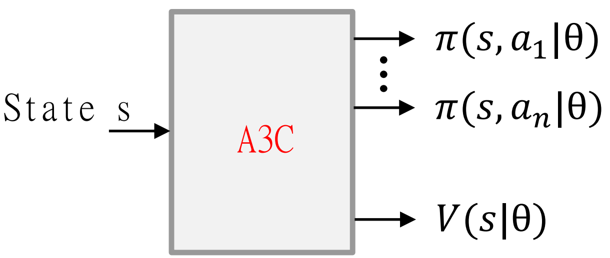
$$Loss = \sum_{s \in S, a \in A} (Q(s_t, a_t|\theta) - Y_t)^2$$

$$Y_t = \left(\sum_{t=0}^{N-1} \gamma^t r_t\right) + r^N V(s_N|\theta)$$



A3C (Asynchronous advantage actor-critic)

- Q-value estimated by n-step sample:
  - $R_t = r_{t+1} + \gamma^1 r_{t+2} + \cdots + \gamma^n V(s_{t+n}|\theta)$
- Actor is updated towards target:
  - $\nabla_\theta J(\theta) \ = \mathrm{E}_{\pi_\theta}[\nabla_\theta \log \pi(s_t, a_t|\theta)\,(R_t - V(s_t|\theta))]$
- Critic is update to minimize MSE:
  - $L(s_t, a_t|\theta) = \left(R_t - V(s_t|\theta)\right)^2$



One-step Actor-Critic (episodic)

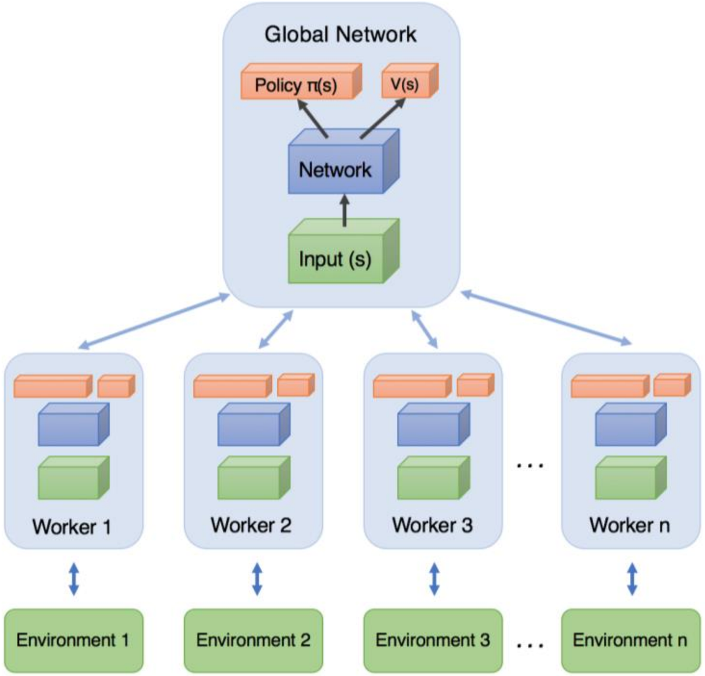
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy weights $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \beta\delta\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha I \delta \nabla_{\boldsymbol{\theta}} \log \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

From: CR-Ko, Deep Reinforcement Learning PG 13.5
https://medium.com/@changrongko/deep-reinforcement-learning-pg-13-5-e3065d5b1dc7

A3C (Asynchronous advantage actor-critic)

- Parallel CPU training.
  - Multiple actor-learners applying online updates in parallel.
    (No experience replay)



- For each worker (Asynchronous part):
  Copy all parameters from the global network.
  keep playing and computing gradients.
  Every N iterations:
    1. Update all gradients to the global network.
    2. Copy all new parameters from the global network.

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t+1$
        $T \leftarrow T+1$
    until terminal s_t **or** $t - t_{start} == t_{max}$

$$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$$

    for $i \in \{t-1, \ldots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$