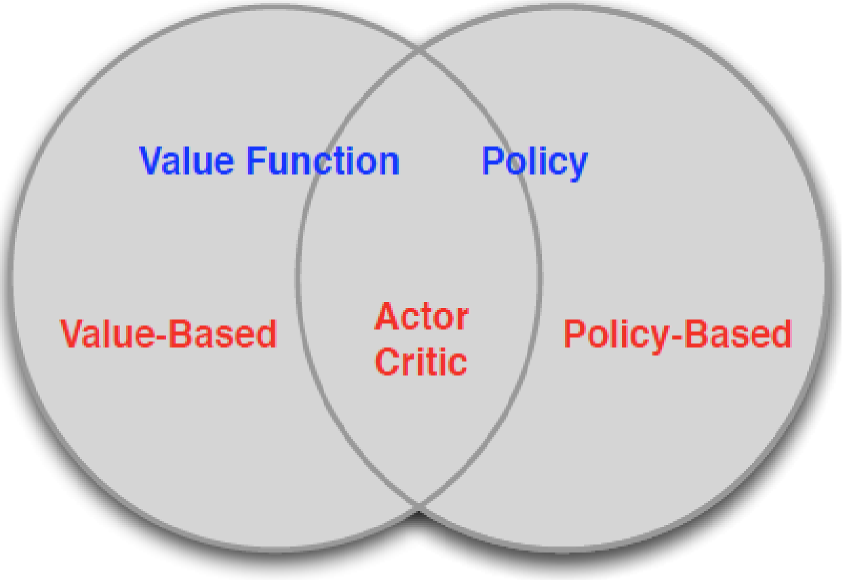
Outline of This Course

- RL1: Introduction to Reinforcement Learning
- RL2: Reinforcement Learning for Lightweight Model
  - Applications
  - Fundamentals of RL
- RL3: Value Based Reinforcement Learning
  - Fundamentals of Value Based RL
  - Algorithms
- RL4: Policy-based Reinforcement Learning
  - Fundamentals of Policy Based RL
  - Algorithms

Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ε-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



References

- DQN
  - Human-level Control Through Deep Reinforcement Learning

    - Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis.
    - DeepMind Technologies
- Double DQN
  - Deep Reinforcement Learning with Double Q-learning

    - Hado van Hasselt, Arthur Guez, David Silver
    - Google DeepMind
- Actor-Critic (discrete action space)
  - D. Silver’s slides (DRL course)
  - 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
- Contributors for the slides include: 蔡承倫, 林九州, 何國豪, etc.

Value-Based Reinforcement Learning

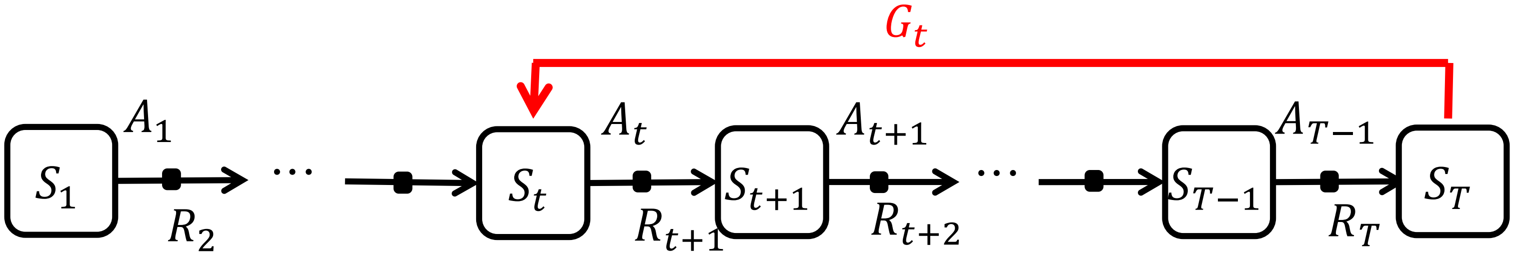
- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

Model Free Reinforcement Learning

- No model
- Learn value function (and/or policy) from experience
- Common Model Free RL
  - Monte-Carlo (MC) Reinforcement Learning
  - Temporal Difference (TD) Reinforcement Learning
  - TD(λ)

Monte-Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is model-free:
  - no knowledge of MDP transitions / rewards
- MC learns from complete episodes:
  - no bootstrapping
- MC uses the simplest possible idea:
  - value = mean return
- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate



Monte-Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

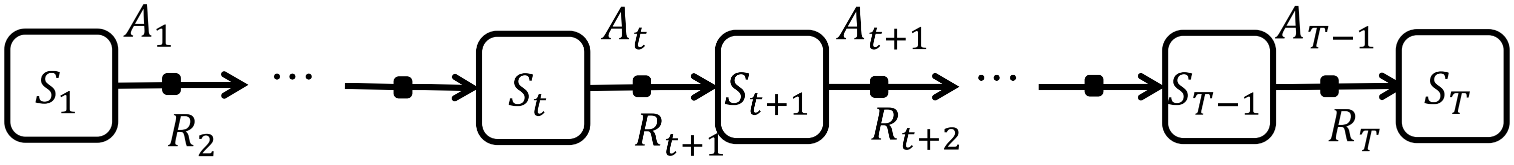
  $S_1, A_1, R_2, \ldots, S_T \sim \pi$
- Recall that the return is the total discounted reward:

  $$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$
- Recall that the value function is the expected return:

  $$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$
- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

Monte-Carlo Policy Evaluation (cont.)

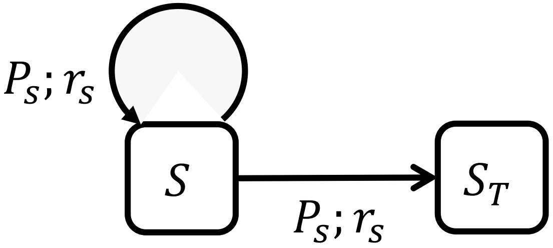
- To evaluate $v_\pi(s)$ at state s
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by mean return $V(s) \leftarrow S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)\ as\ N(s) \rightarrow \infty$



I-Chen Wu

First Visit vs. Every Visit

- To evaluate $v_\pi(s)$ at state s
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by mean return $V(s) \leftarrow S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)\ as\ N(s) \rightarrow \infty$
- What if the same state s is visited in an episode?
  - Do the above for every visit or first visit?
    - What happen for the case in the figure?
    - Both converge quadratically, so this issue is ignored in this course.



Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $\mathrm{x}_1, \mathrm{x}_2, \ldots$ can be computed incrementally,

$$\mu_k = \frac{1}{k} \sum_{j=1}^{k} x_j$$

$$= \frac{1}{k} \left(x_k + \sum_{j=1}^{k} x_j \right)$$

$$= \frac{1}{k} \left(x_k + (k-1)\mu_{k-1} \right)$$

$$= \mu_{k-1} + \frac{1}{k} \left(x_k - \mu_{k-1} \right)$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \ldots, S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

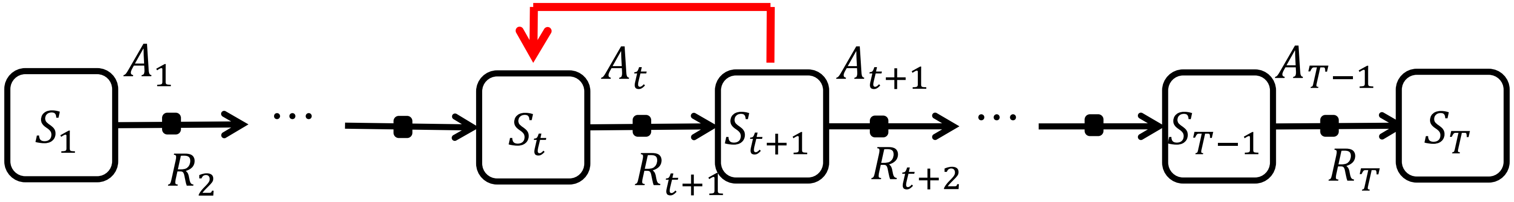
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is model-free:
  - no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes,
  - by bootstrapping
- TD updates a guess towards a guess



MC vs. TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big)$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error

TD vs. MC (I)

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Bias/Variance Trade-Off

- TD target $R_{t+1} + \gamma\, V(S_{t+1})$ is biased estimate of $v_\pi(S_t)$
  - Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is unbiased estimate of $v_\pi(S_t)$
  - True TD target $R_{t+1} + \gamma\, v_\pi(S_{t+1})$ is unbiased estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return:
  - Return depends on many random actions, transitions, rewards
  - TD target depends on only one random action, transition, reward

MC vs. TD (II)

- MC has high variance, zero bias
  - Good convergence properties (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$ (but not always with function approximation)
  - More sensitive to initial value

Batch MC and TD

- MC and TD converge: $V(s) \rightarrow v_\pi(s)$ as experience $\rightarrow \infty$
- But what about batch solution for finite experience?

$$s_1^1, a_1^1, r_2^1, \ldots, s_{T_1}^1$$
$$\vdots$$
$$s_1^k, a_1^k, r_2^k, \ldots, s_{T_k}^k$$

  – e.g. Repeatedly sample episode $k \in [1, K]$
  – Apply MC or TD(0) to episode k

AB Example

Two states A, B; no discounting; 8 episodes of experience

A, 0, B, 0

B, 1

B, 1

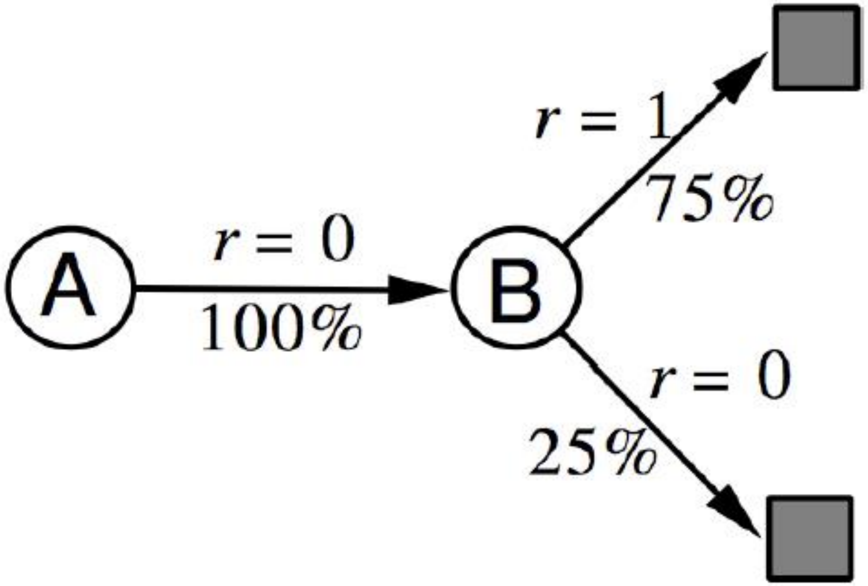
B, 1

B, 1

B, 1

B, 1

B, 0



What is $V(A), V(B)$?

Both MC and TD will obtain different values!!

Certainty Equivalence

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}\left(G_t^k - V(s_t^k)\right)^2$$

  - In the AB example, $V(A) = 0$, $V(B) = 0.75$
- TD(0) converges to solution of max likelihood Markov model
  - Solution to the MDP <$\mathcal{S}$, $\mathcal{A}$, $\hat{\mathcal{P}}$, $\hat{\mathcal{R}}$, γ> that best fits the data

$$\hat{P}_s^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k} 1\left(s_t^k, a_t^k, s_{t+1}^k = s, a, s'\right)$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k} 1\left(s_t^k, a_t^k = s, a\right) r_t^k$$

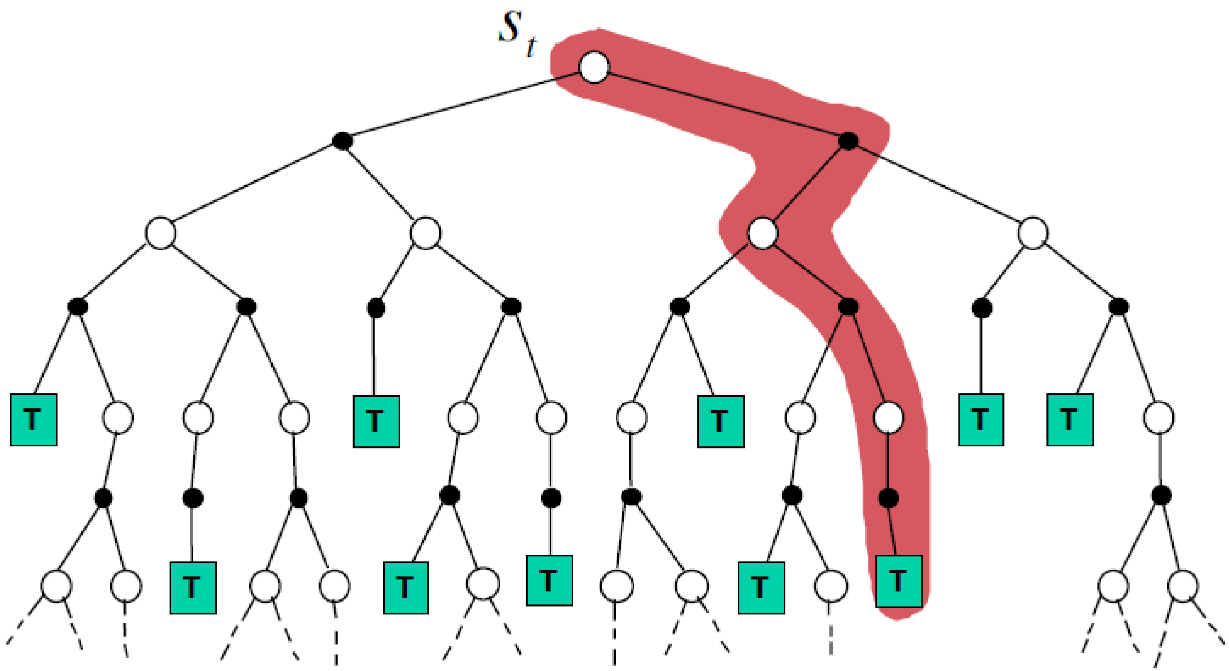
  - In the AB example, $V(A) = 0.75$, $V(B) = 0.75$

MC vs. TD (III)

- TD exploits Markov property
  - Usually more efficient in Markov environments
    - So, TD works well for MDP problems like 2048.
- MC does not exploit Markov property
  - Usually more effective in non-Markov environments
    - MC works fine for non-MDP too.

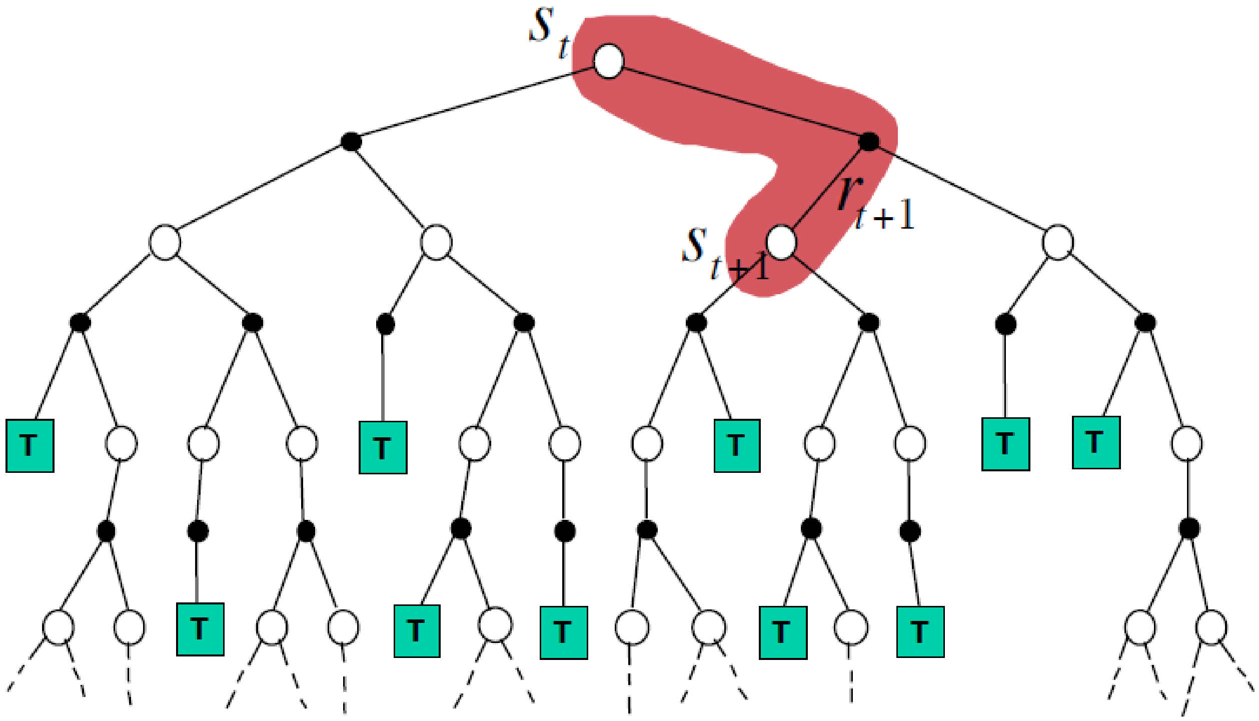
Monte-Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



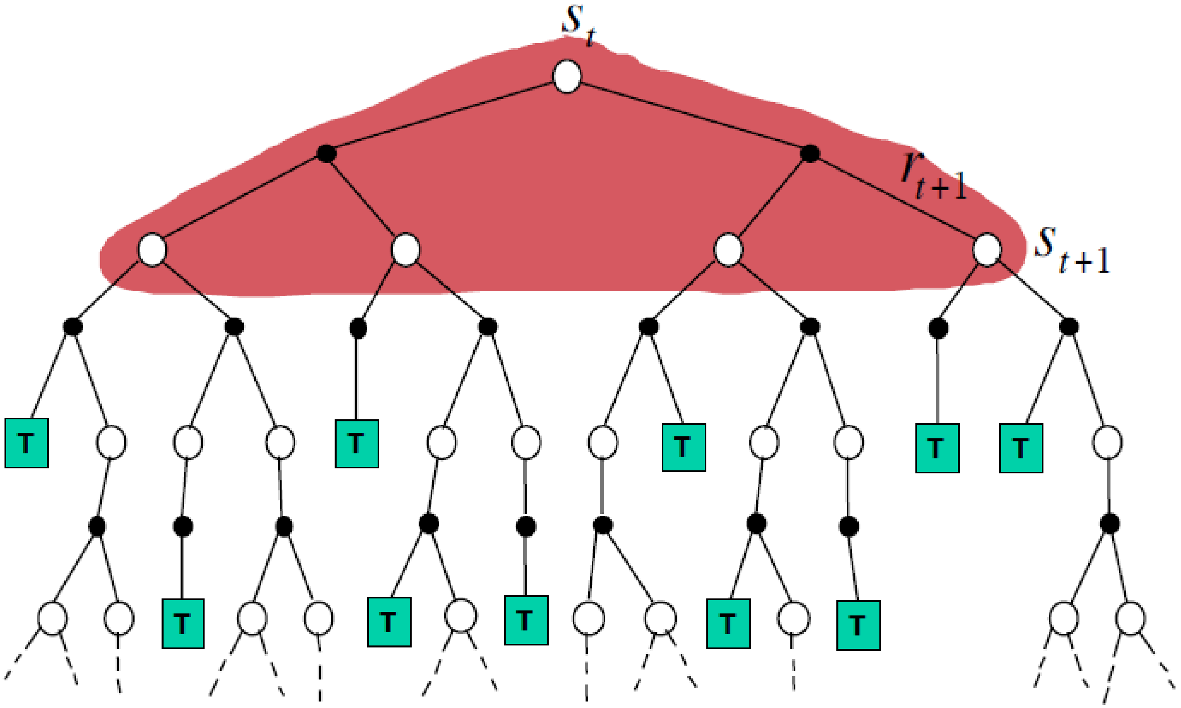
Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



Dynamic Programming Backup

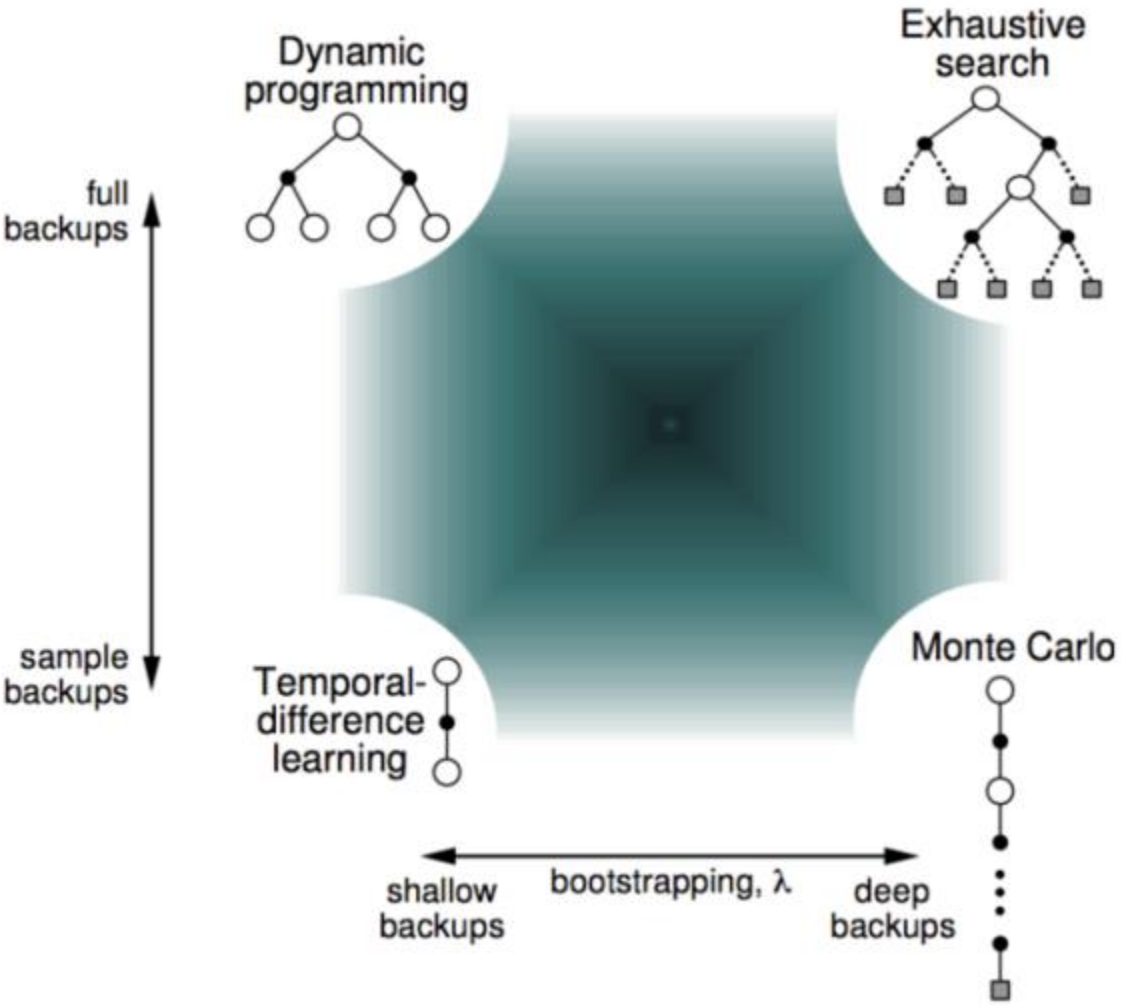
$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$$



Bootstrapping and Sampling

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

Unified View of Reinforcement Learning

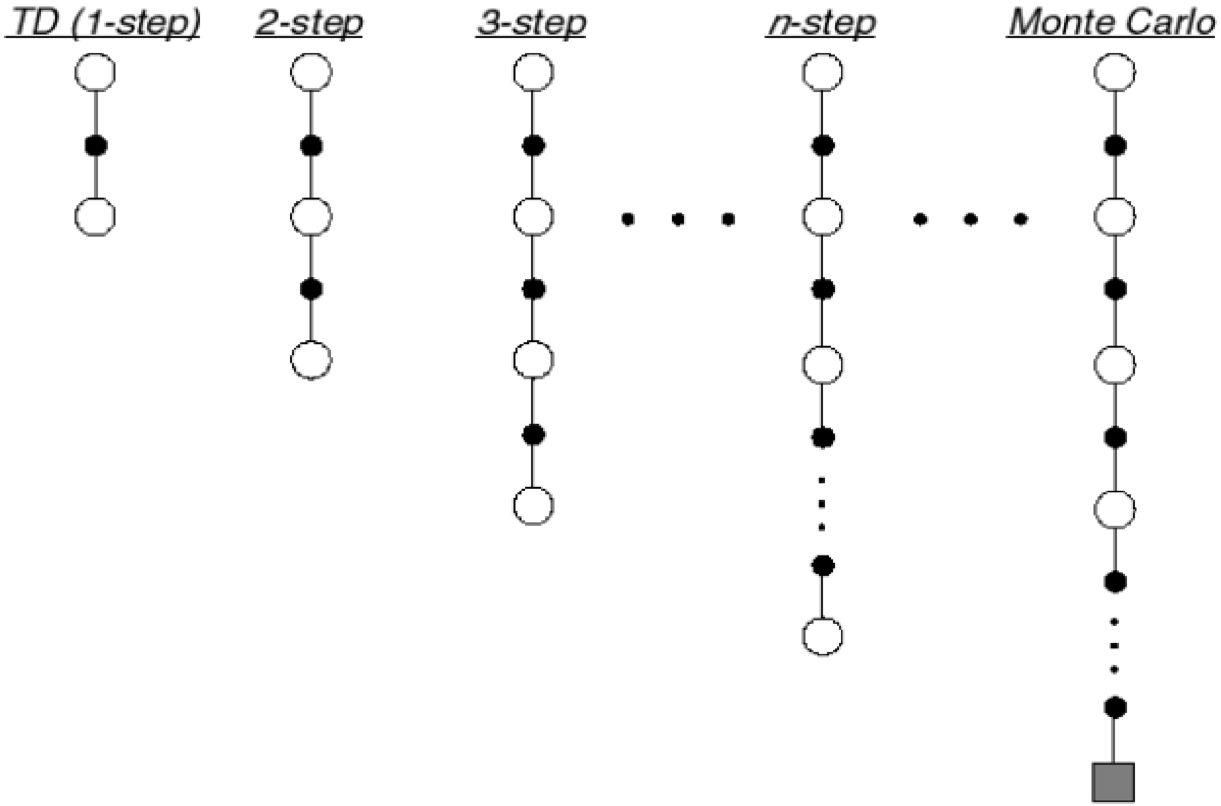


General TD Learning

- Review TD
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$
    $$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
  - For MC learning, the TD target is replaced by G_t
    $$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$
- Question: a more general TD target?
- Investigate TD in a more general manner.
- A typical one: TD(λ)

n-Step Prediction

- Let TD target look n steps into the future



n-Step Return

- Define the n-step return

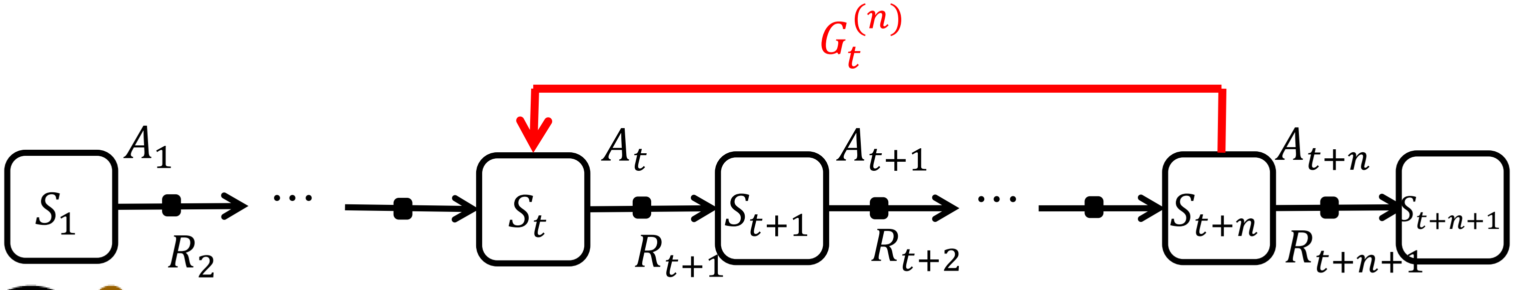
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- Consider the following n-step returns for n $= 1,2,\infty$

$$\begin{array}{ll} \text{n} = 1 & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\ \text{n} = 2 & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\ \vdots & \vdots \\ \text{n} = \infty & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T, \text{ if ends at } T. \end{array}$$

- n-step temporal-difference learning

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$$



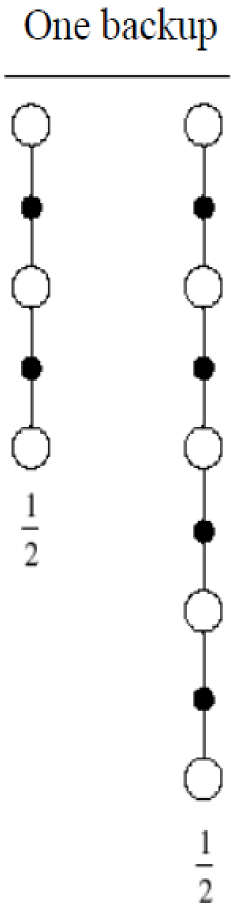
I-Chen Wu

Example of Averaging n-Step Returns

- We can average n-step returns over different n
- Example:
  - average the 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

  - Combines information from two different time-steps
- Next:
  - combine information from all time-steps?



λ-return

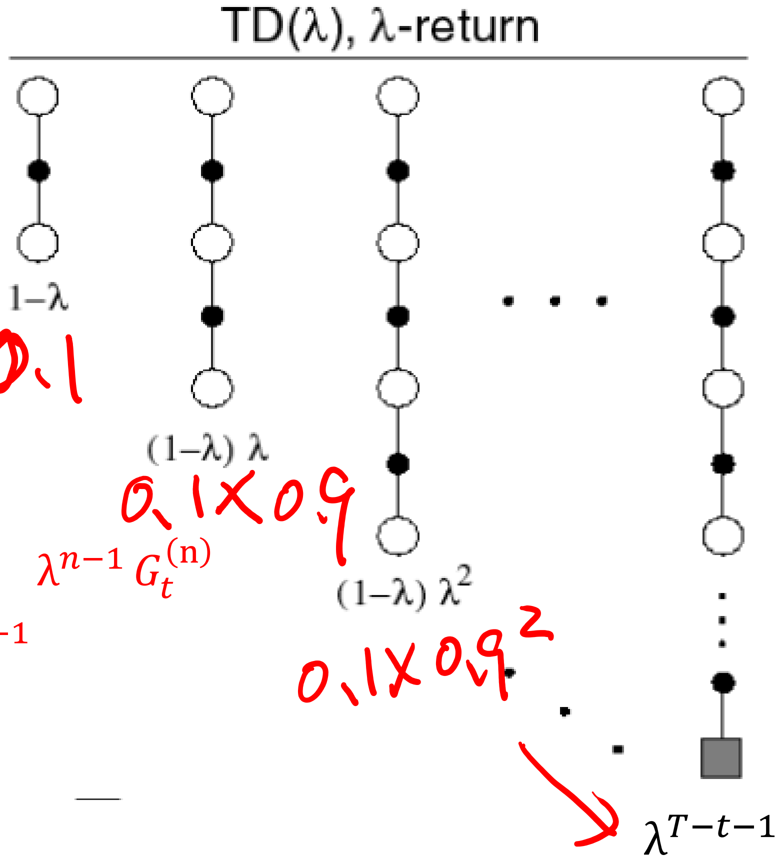
- λ-return G_t^λ:
  - combines all n-step returns $G_t^{(n)}$
- Using weight$(1-\lambda)\,\lambda^{n-1}$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(\mathrm{n})}$$

or (in the case of termination)

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}\, G_t^{(\mathrm{n})} + (1-\lambda)\sum_{n=T-t-1}^{\infty} \lambda^{n-1}\, G_t^{(\mathrm{n})}$$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}\, G_t^{(\mathrm{n})} + \lambda^{T-t-1} G_t$$

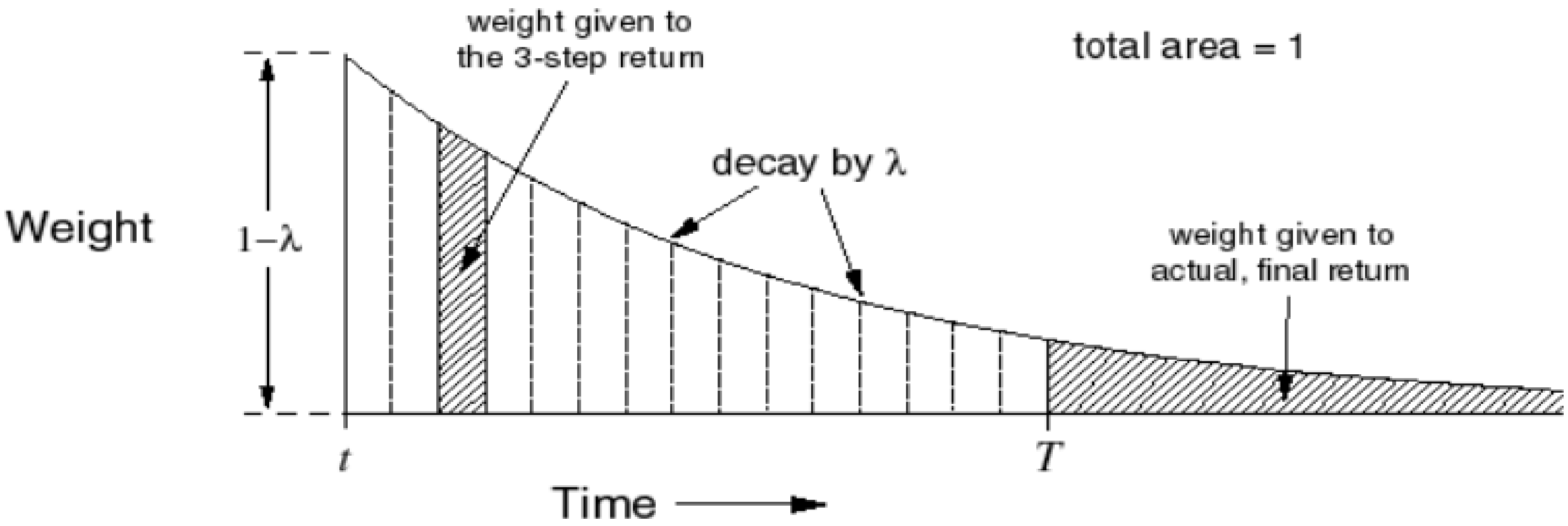


- Forward-view TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

TD(λ) Weighting Function

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$



TD(λ) and TD(0)

- When $\lambda = 0$, only current state is updated

  $$V(s) \leftarrow V(s) + \alpha\delta_t$$

  - This is exactly equivalent to TD(0) update

TD(λ) and MC

- When $\lambda = 0$, only current state is updated, → TD(0)=TD

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(\mathrm{n})} + \lambda^{T-t-1} G_t = G_t^{(1)}$$

  – This is exactly equivalent to TD target.

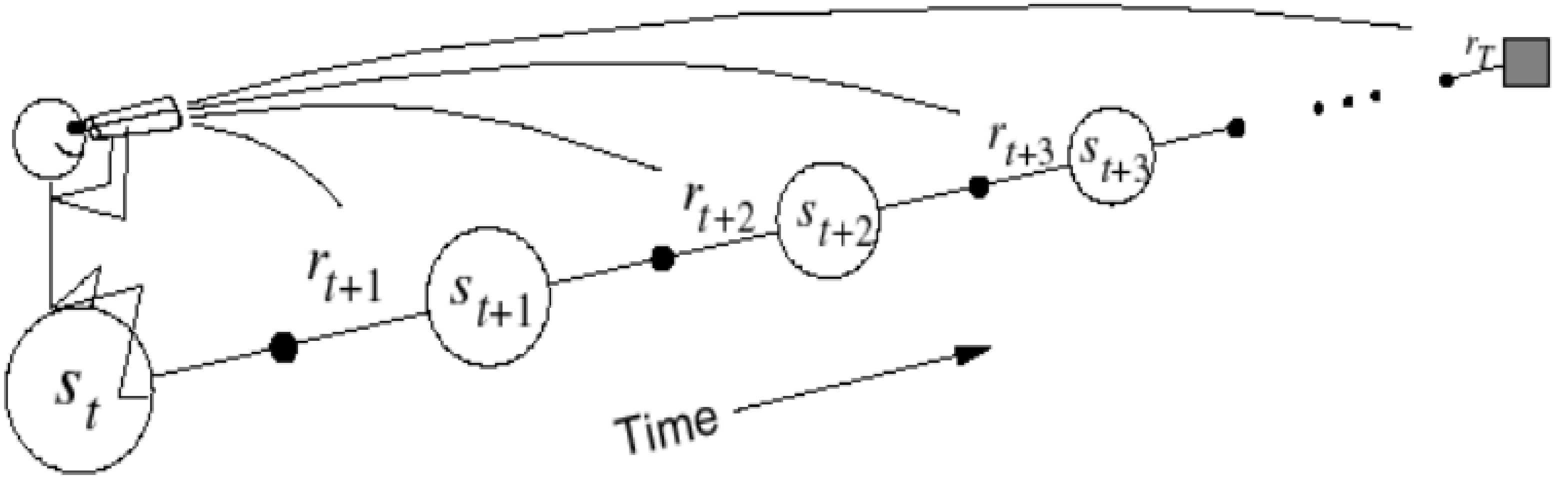
- When $\lambda = 1$, TD(1) = MC

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(\mathrm{n})} + \lambda^{T-t-1} G_t = G_t$$

  – This is exactly equivalent to MC target.

Forward-view TD(λ)

- Update value function towards the λ-return
- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes



Backward View TD(λ)

- Forward view provides theory
- Backward view provides mechanism
  - Update online, every step, from incomplete sequences

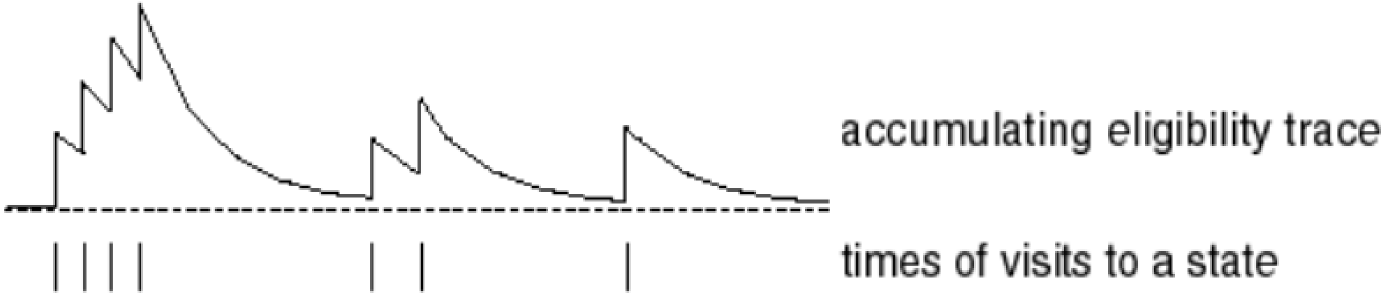
- Notes:
  - You may ignore it now.
  - Consider backward (eligible traces) only when you try to implement it.

Eligibility Traces

- Credit assignment problem: did bell or light cause shock?
- Frequency heuristic: assign credit to most frequent states
- Recency heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

$$E_0(s) = 0$$

$$E_t(s) = \gamma\,\lambda E_{t-1}(s) + 1(S_t = s)$$

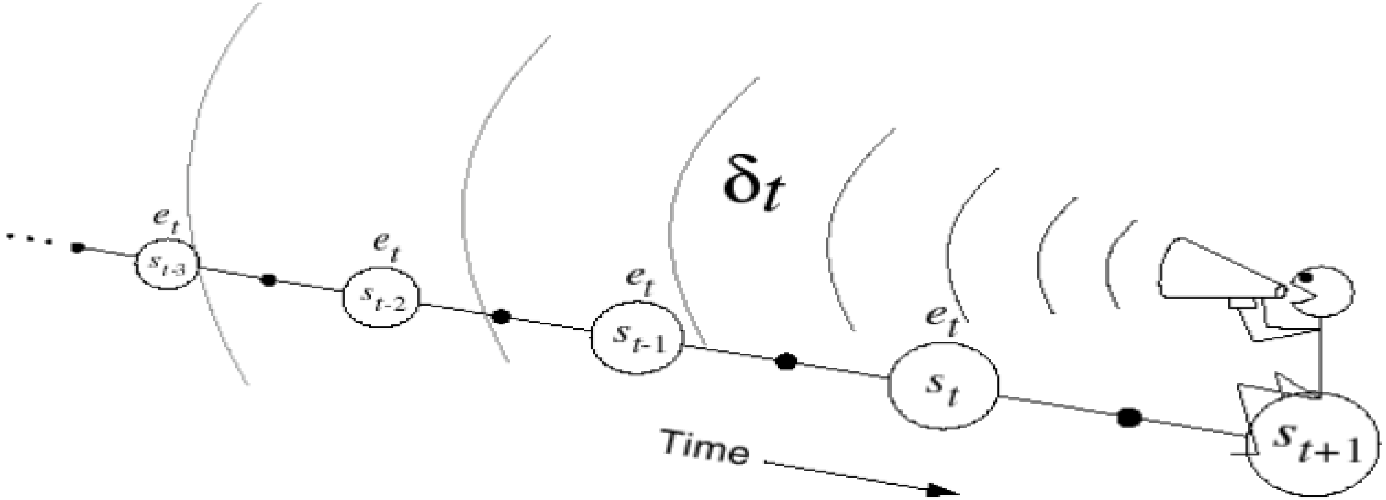


Backward View TD(λ)

- Keep an eligibility trace for every state *s*
- Update value $V(s)$ for every state *s*
- In proportion to TD-error δ_t and eligibility trace $E_t(s)$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$



Eligibility Trace

- Explain more in policy-based reinforcement learning for GAE (Generalized Advantage Estimator).
  - (See Page 37, “Advantages and TD(λ)” in the chapter of policy-based RL.)

Value-Based Reinforcement Learning

- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

Example of Greedy Action Selection

- There are two doors in front of you,
  Always apply the greedy action selection:
  - You open the left door and get reward 0
    $V(left) = 0$
  - You open the right door and get reward +1
    $V(right) = +1$
  - You open the right door and get reward +3
    $V(right) = +2$
  - You open the right door and get reward +2
    $V(right) = +2$
  - $\vdots$
- Are you sure you've chosen the best door?

ε-Greedy Exploration

- ε-greedy policy:

$$\pi(a|s) = \begin{cases} \varepsilon/m + 1 - \varepsilon & if\ a^* = \underset{a\epsilon\mathcal{A}}{\operatorname{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

- Exploration
  - If you always try the best, you don't explore a real better one.
  - With probability ε choose an action at random
    - Simplest idea for ensuring continual exploration
  - All m actions are tried with non-zero probability
- Exploitation
  - If you always choose at random, you don't exploit the best
  - With probability $1 - \varepsilon$ choose the greedy action

ε-Greedy Policy Improvement

(for reference only; can be skipped)

- Theorem
  - For any policy π, the ε-greedy policy π' with respect to q_π is an improvement,$v_{\pi'}(s) \geq v_\pi(s)$
- Proof:

$$v_{\pi'}(s) = \ q_\pi\big(s, \pi'(s)\big) \qquad \text{(follow new policy } \pi' \text{ using old } q_\pi\text{.)}$$
$$= \textstyle\sum_{a\epsilon\mathcal{A}} \pi'(a|s)\, q_\pi(s,a)$$
$$= \frac{\varepsilon}{m}\textstyle\sum_{a\epsilon\mathcal{A}} q_\pi(s,a) + (1-\varepsilon)\max_{a\epsilon A}\ q_\pi(s,a)$$
$$\geq \frac{\varepsilon}{m}\textstyle\sum_{a\epsilon\mathcal{A}} q_\pi(s,a) + (1-\varepsilon)\sum_{a\epsilon\mathcal{A}} \frac{\pi(a|s)-\frac{\varepsilon}{m}}{1-\varepsilon} q_\pi(s,a) \text{ (Lemma)}$$
$$= \frac{\varepsilon}{m}\textstyle\sum_{a\epsilon\mathcal{A}} q_\pi(s,a) + \sum_{a\epsilon\mathcal{A}}(\pi(a|s) - \frac{\varepsilon}{m})\, q_\pi(s,a)$$
$$= \ \textstyle\sum_{a\epsilon\mathcal{A}} \pi(a|s) q_\pi(s,a) = v_\pi(s)$$

- Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

A Lemma in the Previous Proof

(for reference only; can be skipped)

(the sum is a weighted average with nonnegative weights summing to 1, and as such it must be less than or equal to the largest number averaged)

- Lemma: For the previous proof, assume $\pi(a|s) - \frac{\varepsilon}{m} \geq 0$.

$$\max_{a\epsilon A} q_\pi(s,a) \geq \sum_{a\epsilon\mathcal{A}} \frac{\pi(a|s)-\frac{\varepsilon}{m}}{1-\varepsilon} q_\pi(s,a)$$

- Proof: Assume all weights $w_a \geq 0$, and $\sum_{a\epsilon\mathcal{A}} w_a = 1$.

Then, $\max_{a\epsilon A} q_\pi(s,a) \geq \sum_{a\epsilon\mathcal{A}} w_a\, q_\pi(s,a)$

Since weights $\frac{\pi(a|s)-\frac{\varepsilon}{m}}{1-\varepsilon} \geq 0$ and their summation = 1,

we have $\max_{a\epsilon A} q_\pi(s,a) \geq \sum_{a\epsilon\mathcal{A}} \frac{\pi(a|s)-\frac{\varepsilon}{m}}{1-\varepsilon} q_\pi(s,a)$

Key Idea

- Key idea:
  - $\pi(a|s) - \frac{\varepsilon}{m}$ is non-negative, as long as ε is monotonically decreasing.
- Example:
  Assume ε = 0.4 and m=4 (4 actions, a_1, a_2, a_3, a_4).
  - $\pi(a_1|s) = 0.4$, and $q_\pi(a_1|s) = 20$
  - $\pi(a_2|s) = 0.3$, and $q_\pi(a_2|s) = 30$ (max in the new policy π')
  - $\pi(a_3|s) = 0.2$, and $q_\pi(a_3|s) = 15$
  - $\pi(a_4|s) = 0.1$, and $q_\pi(a_4|s) = 15$
  - Works when ε remains the same or drops to a smaller number, say 0.3.

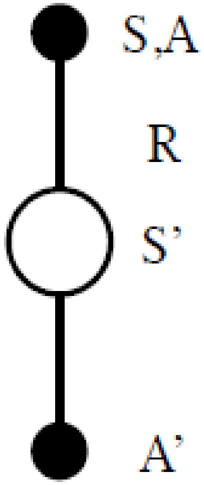
Value-Based Reinforcement Learning

- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

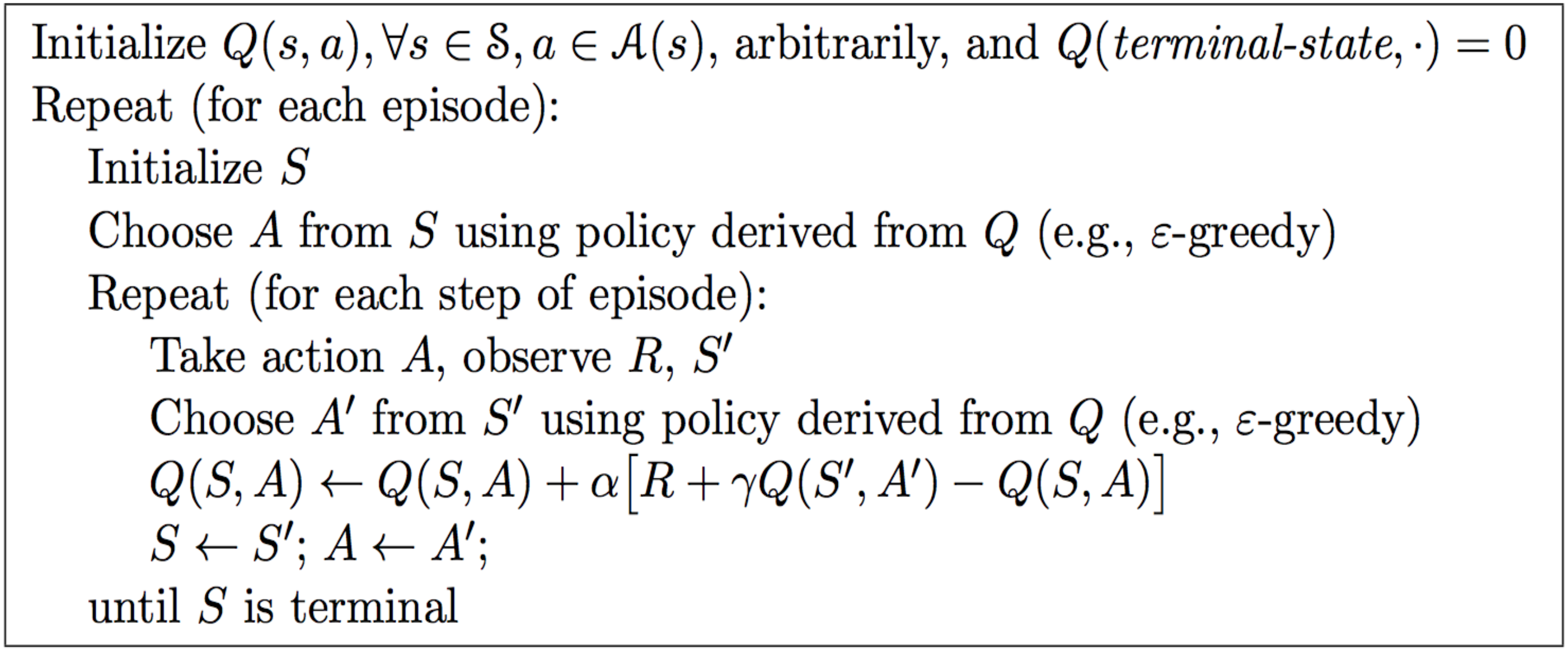
Updating Action-Value Functions with Sarsa

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

- Notice: Interesting naming



Sarsa Algorithm for On-Policy Control



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

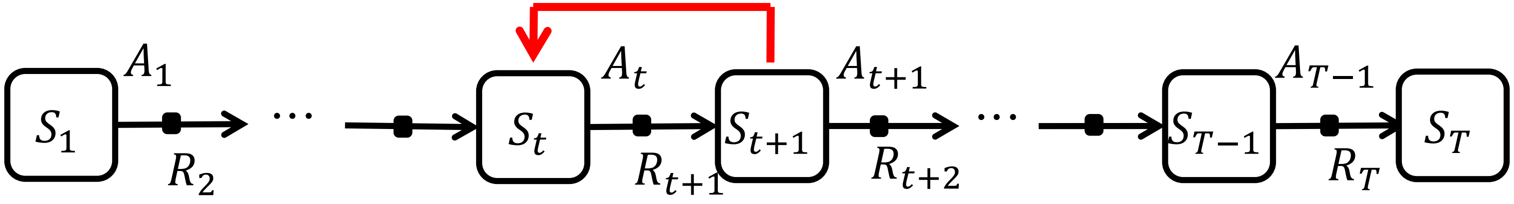
- Sarsa converges to the optimal action-value function
- n-step Sarsa – like n-step return
- Sarsa(λ) – like TD(λ)

Off-Policy Learning

- Evaluate current policy $\pi(a|s)$ to compute $V_\pi(s)$ or $q_\pi(s,a)$, while following an old policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$$

- Why is this important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{t-1}$
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy



Current Policy π

Importance Sampling

- Estimate the expectation of a different distribution

$$\mathbb{E}_{X\sim P}[f(X)]$$
$$= \sum P(X) f(X)$$
$$= \sum Q(X) \frac{P(X)}{Q(X)} f(X)$$
$$= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)} f(X)\right]$$

Importance Sampling for Off-Policy Monte-Carlo

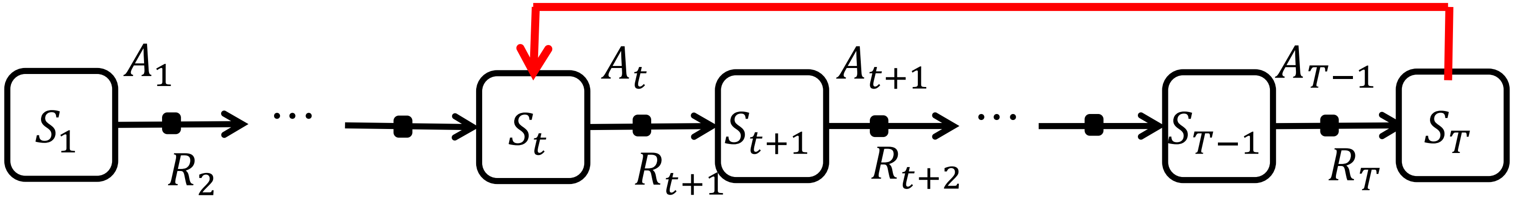
- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1})}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1})} \dots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards corrected return

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\pi/\mu} - V(S_t)\right)$$

- Cannot use if μ is zero when π is non-zero
- Importance sampling can dramatically increase variance



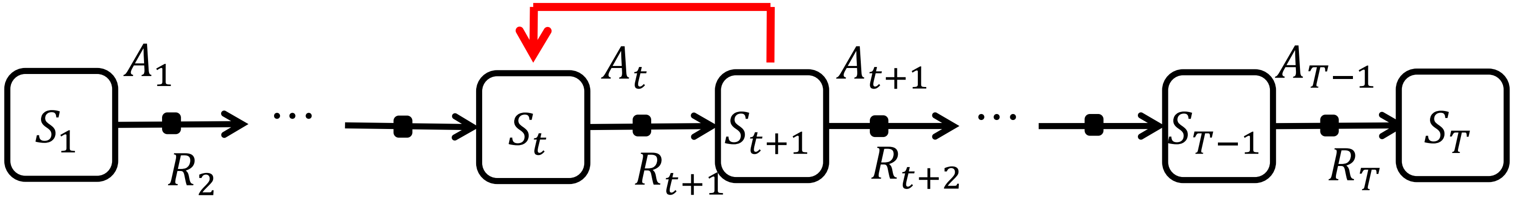
Current Policy π

Importance Sampling for Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target $R+\gamma V(S')$ by importance sampling
- Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}(R_{t+1}+\gamma V(S_{t+1})) - V(S_t)\right)$$

- Much lower variance than Monte-Carlo importance sampling (since just one step)
- Policies only need to be similar over a single step



Current Policy π

Q-Learning

- We now consider off-policy learning of action-values $Q(s,a)$
- No importance sampling is required
- Next action is chosen using the old policy

  $$A_{t+1} \sim \mu(\cdot \mid S_{t+1})$$

- But we consider alternative successor action

  $$A' \sim \pi(\cdot \mid S_{t+1})$$

- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

Off-Policy Control with Q-Learning

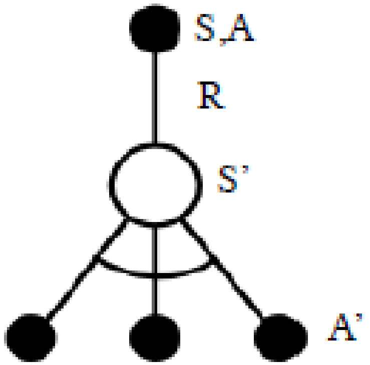
- We now allow both old and current policies to improve
- The current policy π is greedy w.r.t. $Q(s,a)$

$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')$$

- The old policy μ is e.g. ε-greedy w.r.t. $Q(s,a)$
- The Q-learning target then simplifies:

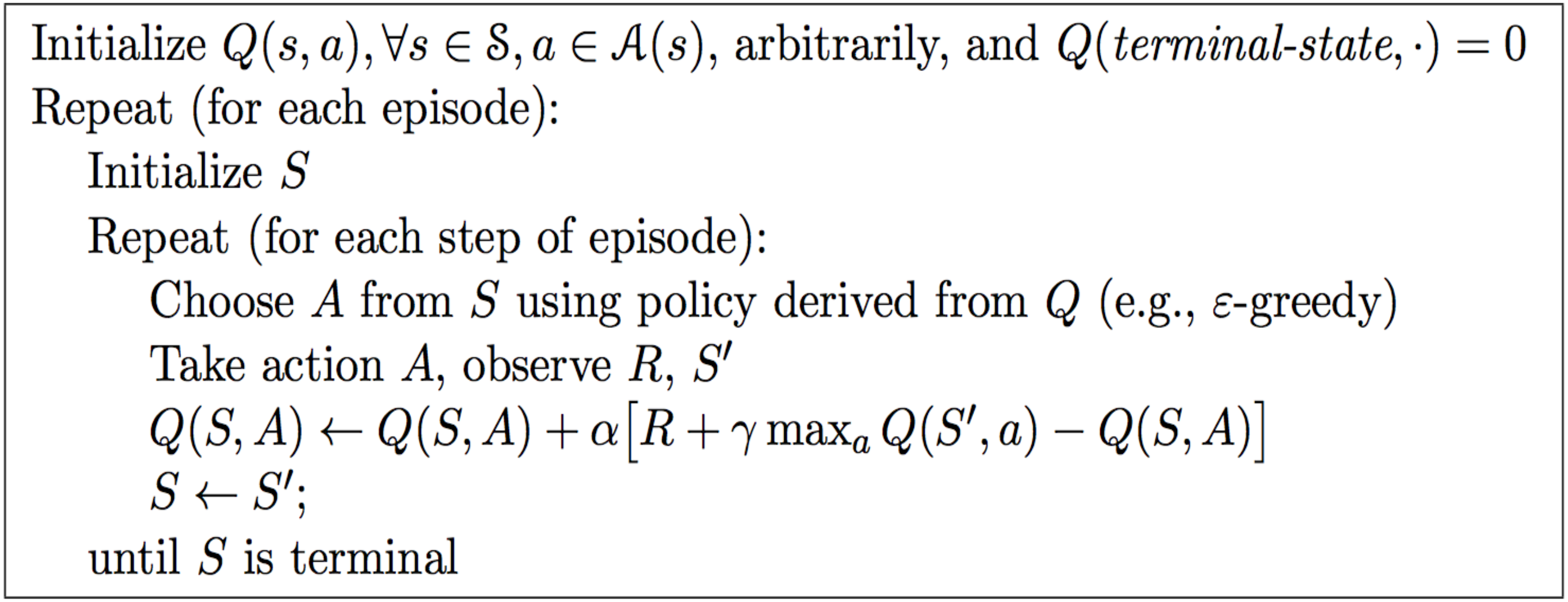
$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ & = R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')\right) \\ & = R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Q-Learning Control Algorithm



- $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$
- Theorem
  - Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow q_*(s,a)$

Q-Learning Algorithm for Off-Policy Control



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

Value-Based Reinforcement Learning

- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

Large-Scale Reinforcement Learning

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space

- How can we scale up the model-free methods for prediction and control from the last two lectures?

Value Function Approximation

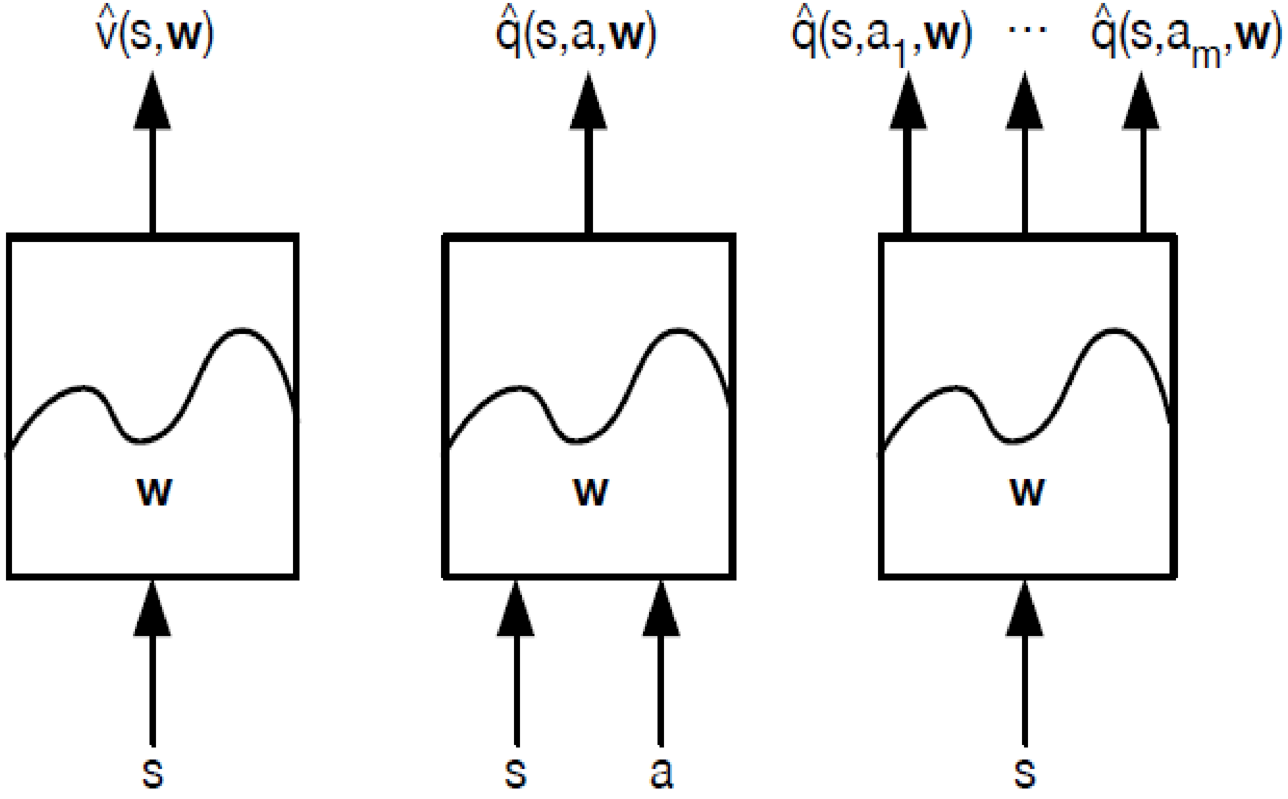
- So far we have represented value function by a lookup table
  - Every state s has an entry $V(s)$
  - Or every state-action pair s; a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

$$\hat{v}(s, w) \approx v_\pi(s)$$

$$\text{or } \hat{q}(s, a, w) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states
  - Update parameter w using MC or TD learning

Types of Value Function Approximation


$\hat{v}(s,\mathbf{w})$
$\hat{q}(s,a,\mathbf{w})$
$\hat{q}(s,a_1,\mathbf{w})$
$\cdots$
$\hat{q}(s,a_m,\mathbf{w})$
$\mathbf{w}$
$\mathbf{w}$
$\mathbf{w}$
s
s
a
s

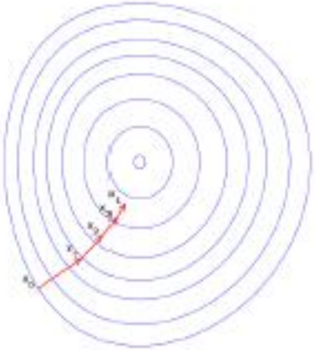
Which Function Approximator?

- There are many function approximators, e.g.
  - Linear combinations of features
  - Neural network
  - Decision tree
  - Nearest neighbour
  - Fourier / wavelet bases
  - ...
- Better to consider differentiable function approximators (in red above)
- Furthermore, we require a training method that is suitable for non-stationary, non-iid data

Gradient Descent

- Let $J(w)$ be a differentiable function of parameter vector w
- Define the gradient of $J(w)$ to be

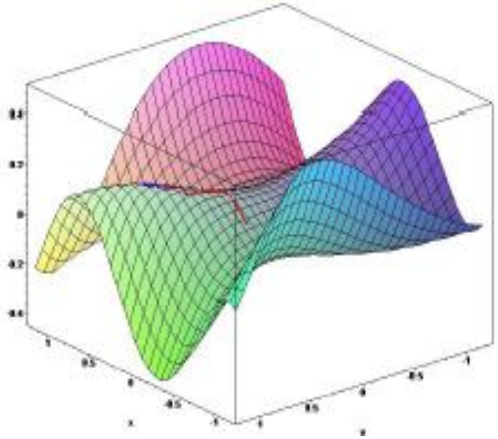
$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$



- To find a local minimum of $J(w)$
- Adjust w in direction of -ve gradient

$$\Delta \mathrm{w} = -\frac{1}{2}\alpha \nabla_w J(w)$$

  - where α is a step-size parameter



I-Chen Wu

Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector *w*
  - minimizing mean-squared error between approximate value function $\hat{v}(s, w)$ and true value function $v_\pi(s)$

$$J(w) = \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S, w))^2]$$

- Gradient descent finds a local minimum

$$\Delta \mathrm{w} = -\frac{1}{2}\alpha \nabla_w J(w)$$

$$= \alpha \mathbb{E}_\pi[\big(v_\pi(S) - \hat{v}(S, w)\big)\nabla_w \hat{v}(S, w)]$$

- Stochastic gradient descent samples the gradient

$$\Delta \mathrm{w} = \ \alpha\big(v_\pi(S) - \hat{v}(S, w)\big)\nabla_w \hat{v}(S, w)$$

- Expected update is equal to full gradient update

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j$$

- Objective function is quadratic in parameters *w*

$$J(w) = \mathbb{E}_\pi[(v_\pi(S) - x(S)^T w)^2]$$

- Stochastic gradient descent converges on global optimum
- Update rule is particularly simple

$$\nabla_w \hat{v}(S, w) = x(S)$$

$$\Delta \mathrm{w} = \alpha\big(v_\pi(S) - \hat{v}(S, w)\big)x(S)$$

- Update = step-size × prediction error × feature value

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a target for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathrm{w} = \alpha\big(G_t - \hat{v}(S_t, w)\big)\nabla_w \hat{v}(S_t w)$$

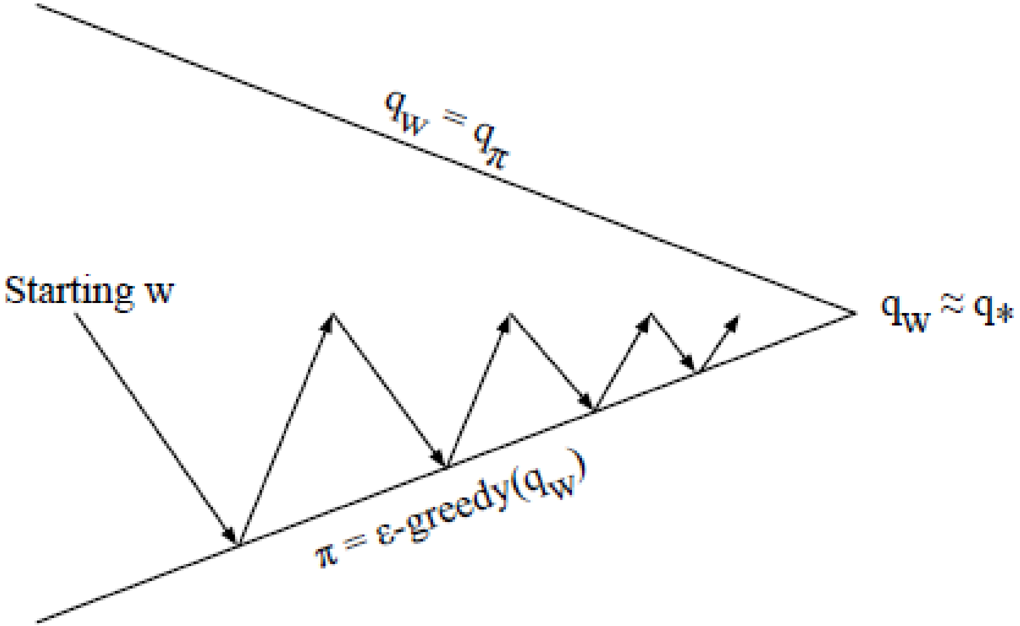
  - For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$

$$\Delta \mathrm{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t w)$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathrm{w} = \alpha(G_t^\lambda - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t w)$$

Control with Value Function Approximation



- Policy evaluation
  - Approximate policy evaluation, $\hat{q}(\cdot,\cdot,w) \approx q_\pi$
- Policy improvement
  - ε-greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, w) \approx q_\pi(S, A)$$

- Minimize mean-squared error between approximate action-value function $\hat{q}(S, A, w)$ and true action-value function $q_\pi(S, A)$

$$J(w) = \mathbb{E}_\pi[(q_\pi(S, A) - \hat{q}(S, A, w))^2]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_w J(w) = \left(q_\pi(S, A) - \hat{q}(S, A, w)\right)\nabla_w \hat{q}(S, A, w)$$

$$\Delta \mathrm{w} = \ \alpha\left(q_\pi(S, A) - \hat{q}(S, A, w)\right)\nabla_w \hat{q}(S, A, w)$$

Incremental Control Algorithms

- Like prediction, we must substitute a target for $q_\pi(S, A)$
  - For MC, the target is the return G_t

    $$\Delta \mathrm{w} = \alpha\big(G_t + \hat{q}(S_t, A_t, w)\big)\nabla_w \hat{q}(S_t, A_t, w)$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma \mathrm{Q}(S_{t+1}, A_{t+1})$

    $$\Delta \mathrm{w} = \alpha\big(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w)\big)$$

    $$\nabla_w \hat{q}(S_t, A_t, w)$$

  - For forward-view TD(λ), target is the action-value λ-return

    $$\Delta \mathrm{w} = \alpha\big(q_t^\lambda - \hat{q}(S_t, A_t, w)\big)\nabla_w \hat{q}(S_t, A_t, w)$$

  - For backward-view TD(λ), equivalent update is

    $$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w)$$

    $$E_t = \gamma \lambda E_{t-1} + \nabla_w \hat{q}(S_t, A_t, w)$$

    $$\Delta \mathrm{w} = \alpha \delta_t E_t$$

Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is not sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent's experience (“training data”)

Least Squares Prediction

- Given value function approximation $\hat{v}(s, w) \approx v_\pi(s)$
- And experience D consisting of $\langle \text{state}, \text{value} \rangle$ pairs

$$D = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \dots, \langle s_T, v_T^\pi \rangle\}$$

- Which parameters w give the best fitting value fn $\hat{v}(s, w)$?
- Least squares algorithms find parameter vector w minimizing sum-squared error between $\hat{v}(s_t, w)$ and target values v_t^π,

$$LS(w) = \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, w))^2$$

$$= \mathbb{E}_D[(v^\pi - \hat{v}(s, w))^2]$$

Stochastic Gradient Descent with Experience Replay

- Given experience consisting of $\langle \text{state}, \text{value} \rangle$ pairs

$$D = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \dots \langle s_T, v_T^\pi \rangle\}$$

- Repeat:
  - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim D$$

  - Apply stochastic gradient descent update

$$\Delta \mathrm{w} = \alpha(v^\pi - \hat{v}(s, w))\, \nabla_w \hat{v}(s, w)$$

- Converges to least squares solution

$$\mathrm{w}^\pi = \underset{w}{\operatorname{argmin}}\, LS(\mathrm{w})$$

  - Similar for action value function q^π

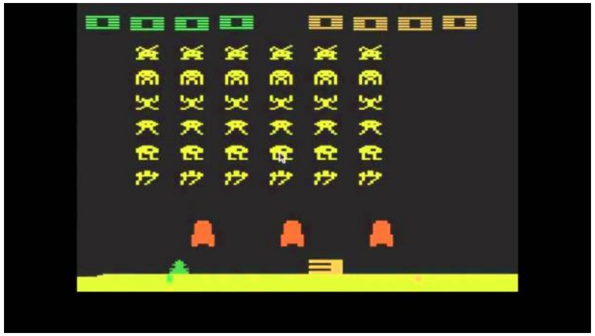
Value-Based Reinforcement Learning

- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

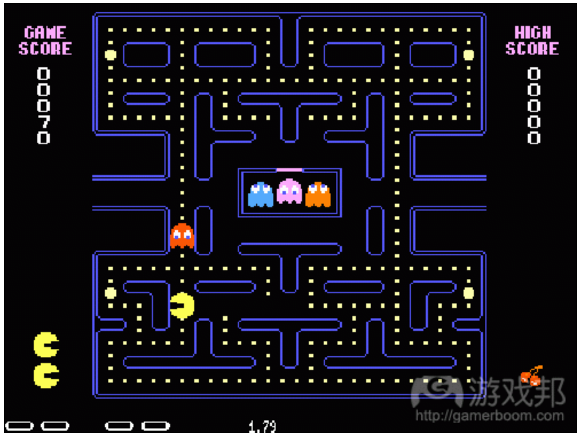
Atari 2600 Games – a Big Success of DQN

- Learn to play Atari games from video only (without knowing the game a priori) by DeepMind, 2013. (in Nature, 2015)

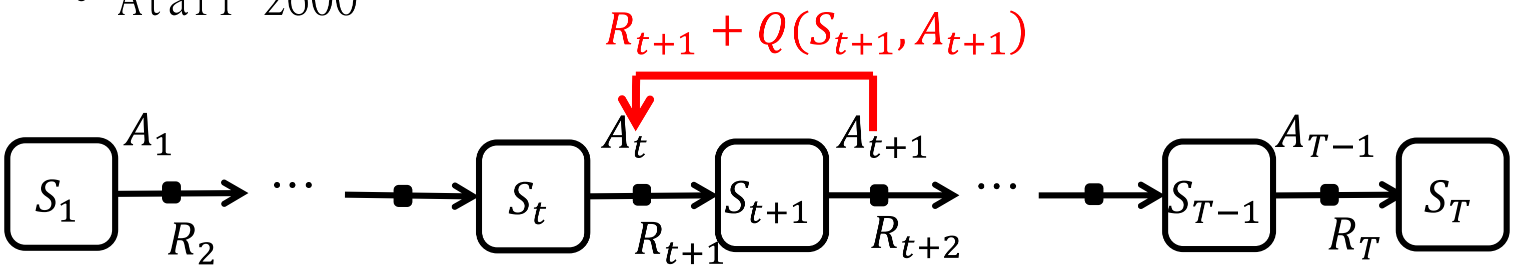
- Atari 2600



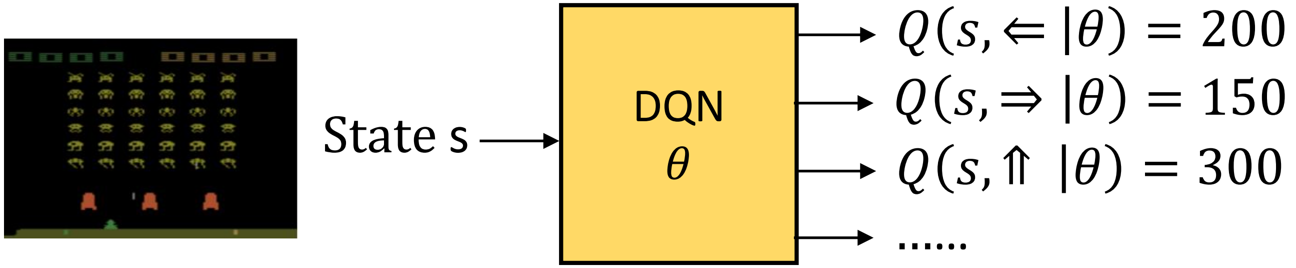
- Space Invader



- PacMan



Illustrations of DQN



State s

DQN

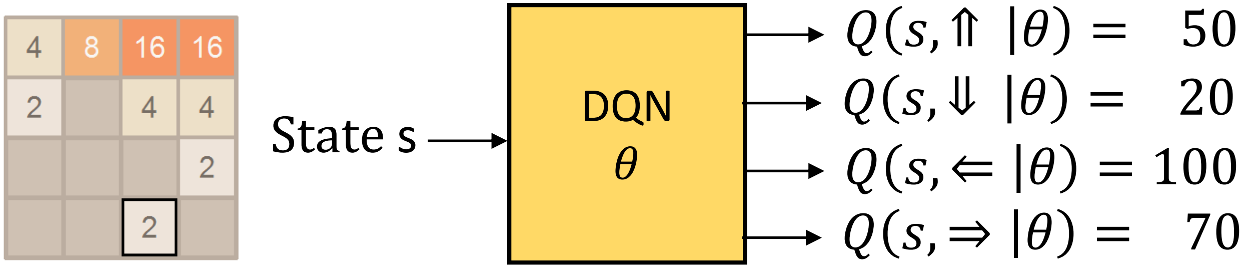
θ

$Q(s, \Leftarrow |\theta) = 200$

$Q(s, \Rightarrow |\theta) = 150$

$Q(s, \Uparrow |\theta) = 300$

……



4	8	16	16
2		4	4
			2
		2	

State s

DQN

θ

$Q(s, \Uparrow |\theta) = 50$

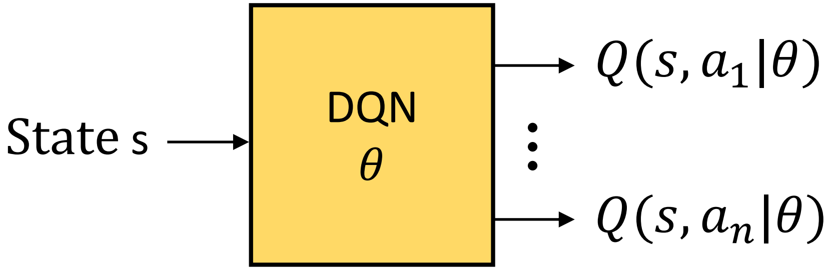
$Q(s, \Downarrow |\theta) = 20$

$Q(s, \Leftarrow |\theta) = 100$

$Q(s, \Rightarrow |\theta) = 70$

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:
  - $\nabla_\theta L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right) \nabla_\theta Q(s_t, a_t|\theta)$

Stability Issues with Deep RL

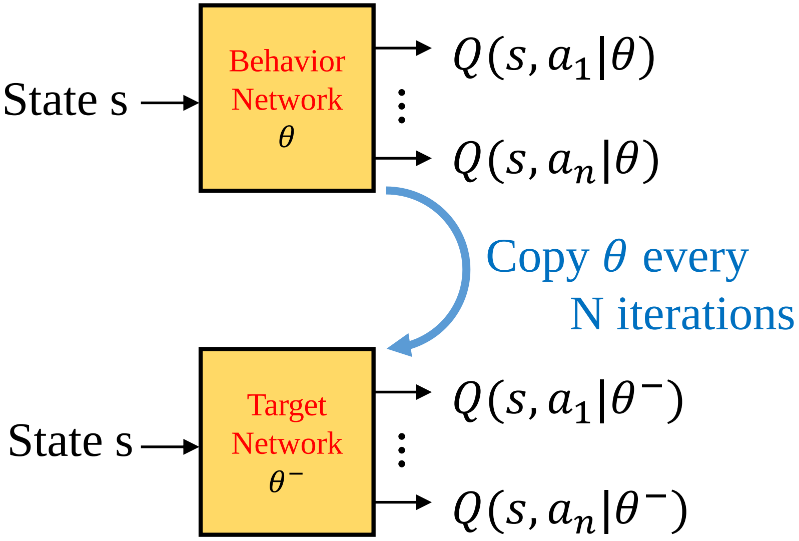
- Data is sequential (overfitting)
  - Successive samples are correlated, non-iid
- Policy changes rapidly with slight changes to Q-values (hard to converge)
  - Policy may oscillate
  - Distribution of data can swing from one extreme to another
- Scale of rewards and Q-values is unknown (not normalized)
  - Naive Q-learning gradients can be large and unstable when backpropagated

Solution for Stability

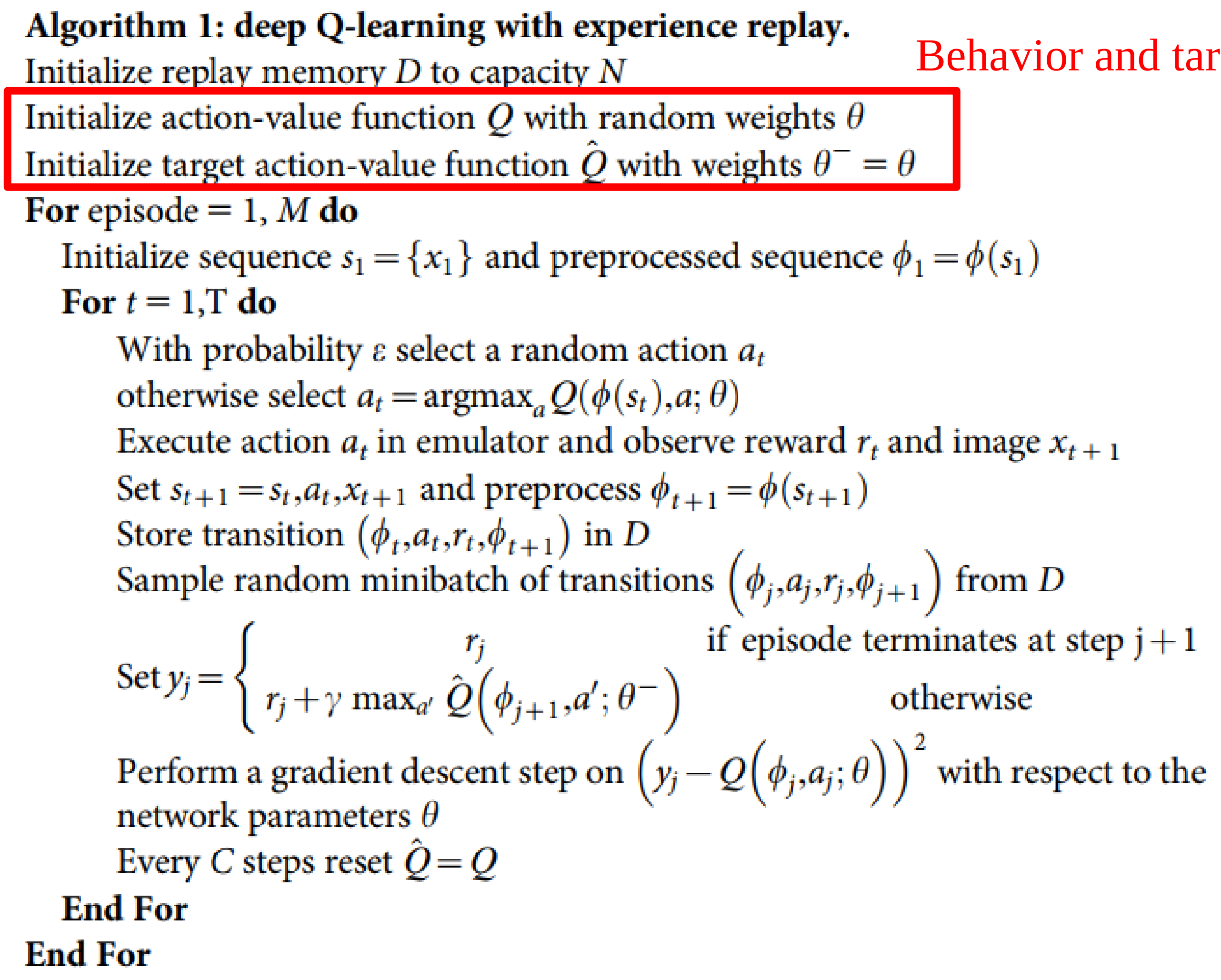
- Use experience replay
  - Break correlations in data, bring us back to iid setting
  - Learn from all past policies
- Freeze target Q-network
  - Avoid oscillations
  - Break correlations between Q-network and target
- Clip rewards or normalize network adaptively to sensible range (simply normalization, not discussed here)
  - Robust gradients

Deep Q Network (DQN)

- Techniques
  1. Target Network with parameters θ^-
  2. Experience Replay
     - Sample experiences at random.

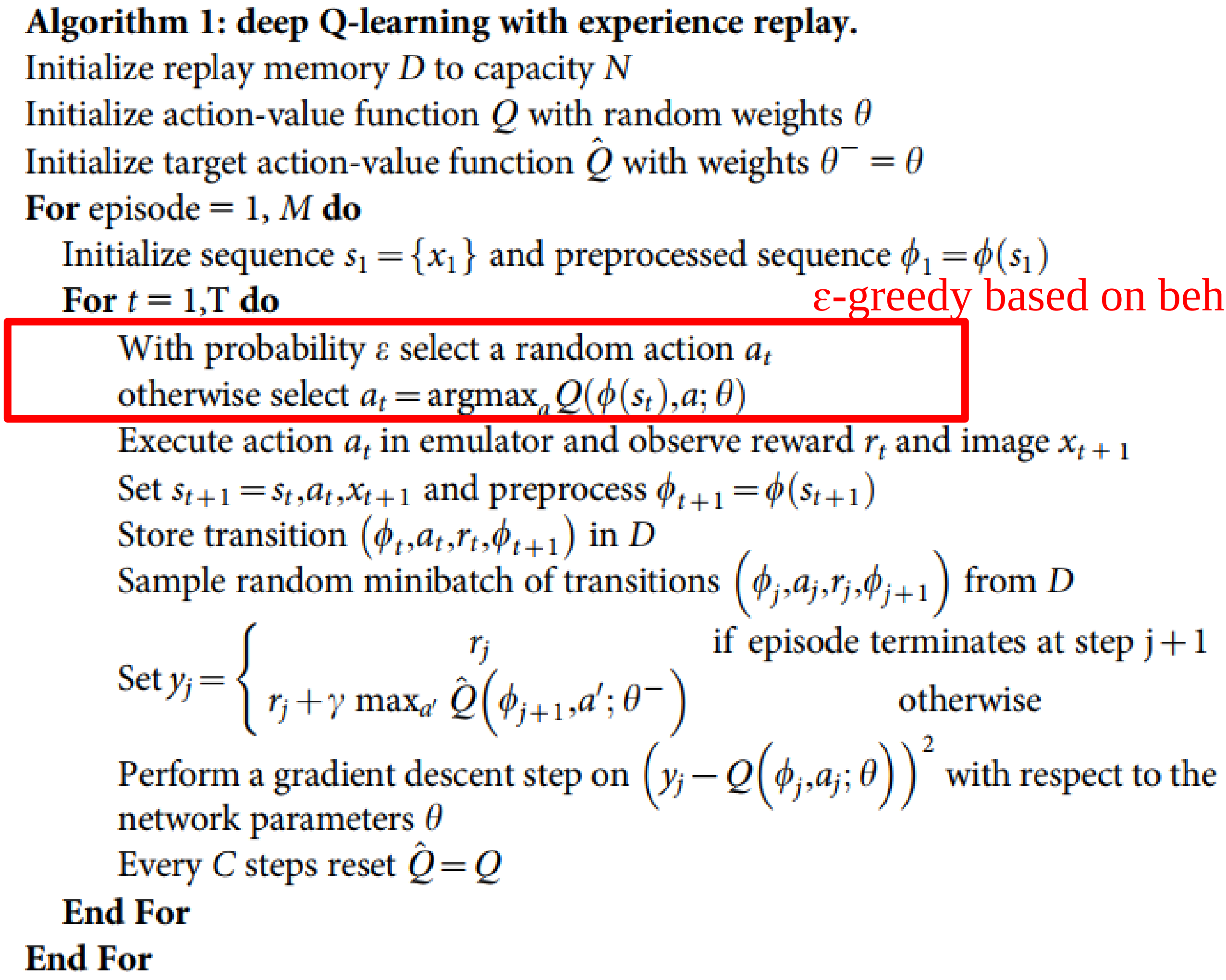


- Apply Target Network on DQN:
  - $Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'|\theta^-)$
  - Gradient descent on behavior network:
    - $\nabla_\theta L_Q(s_t, a_t|\theta) = \left(Y_t^Q - Q(s_t, a_t|\theta)\right) \nabla_\theta Q(s_y, a_t|\theta)$
  - Copy parameters from θ to θ^- every N iterations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



Behavior and target network

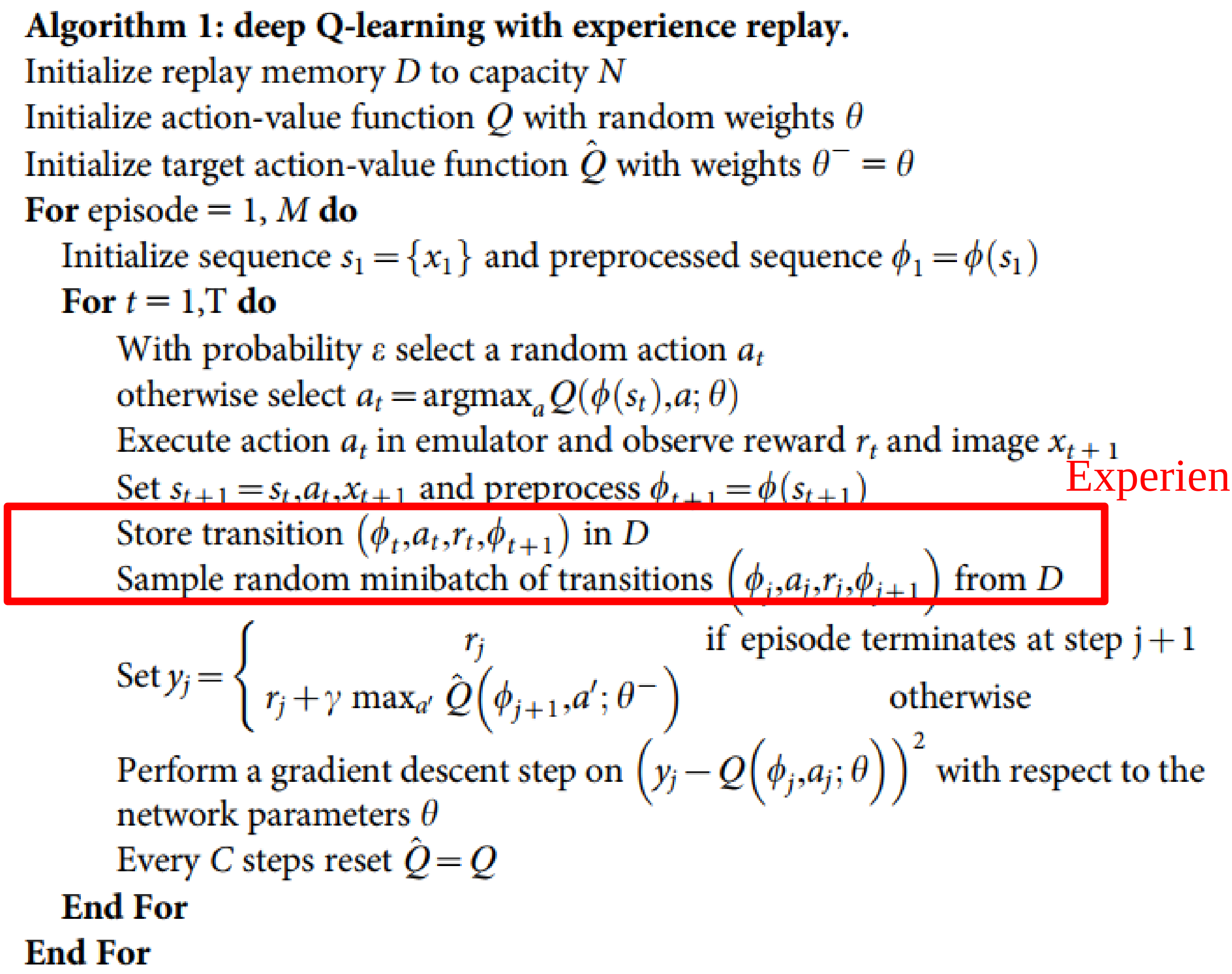
```
**Algorithm 1: deep Q-learning with experience replay.**
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
**For** episode = 1, $M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    **For** $t$ = 1,T **do**
        With probability $\varepsilon$ 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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    **End For**
**End For**
```



Algorithm 1: deep Q-learning with experience replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

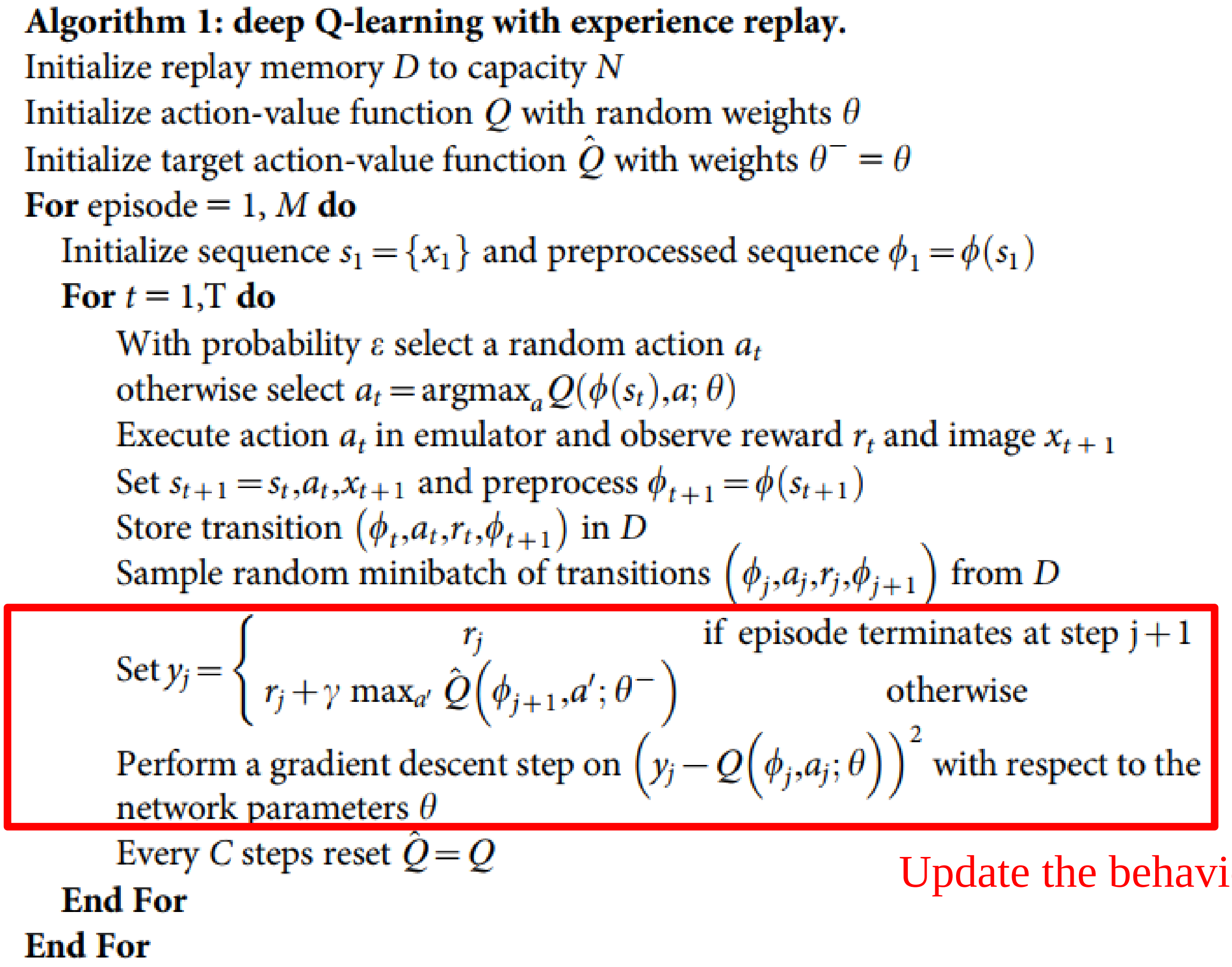
ε-greedy based on behavior network



Algorithm 1: deep Q-learning with experience replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Experience replay



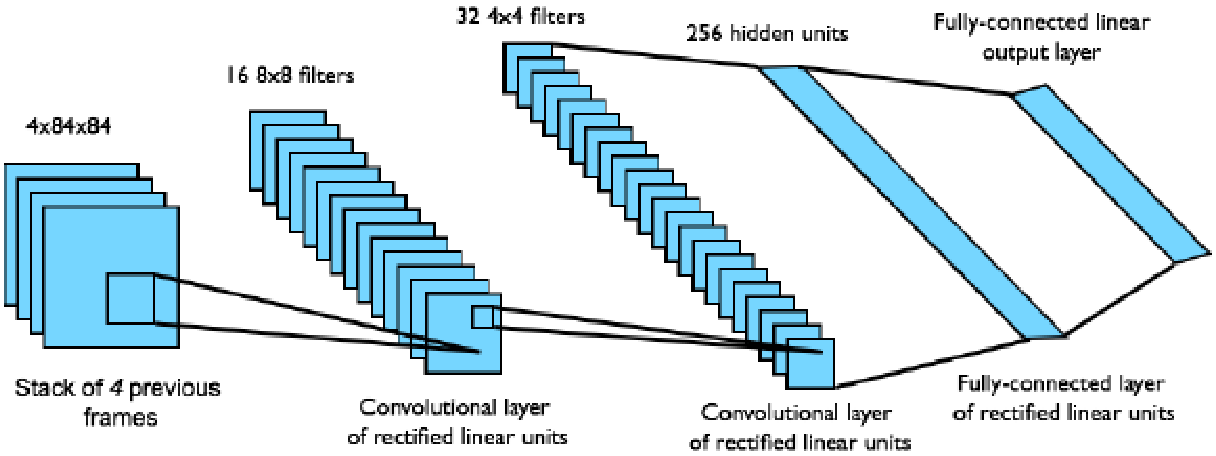
Algorithm 1: deep Q-learning with experience replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**
```

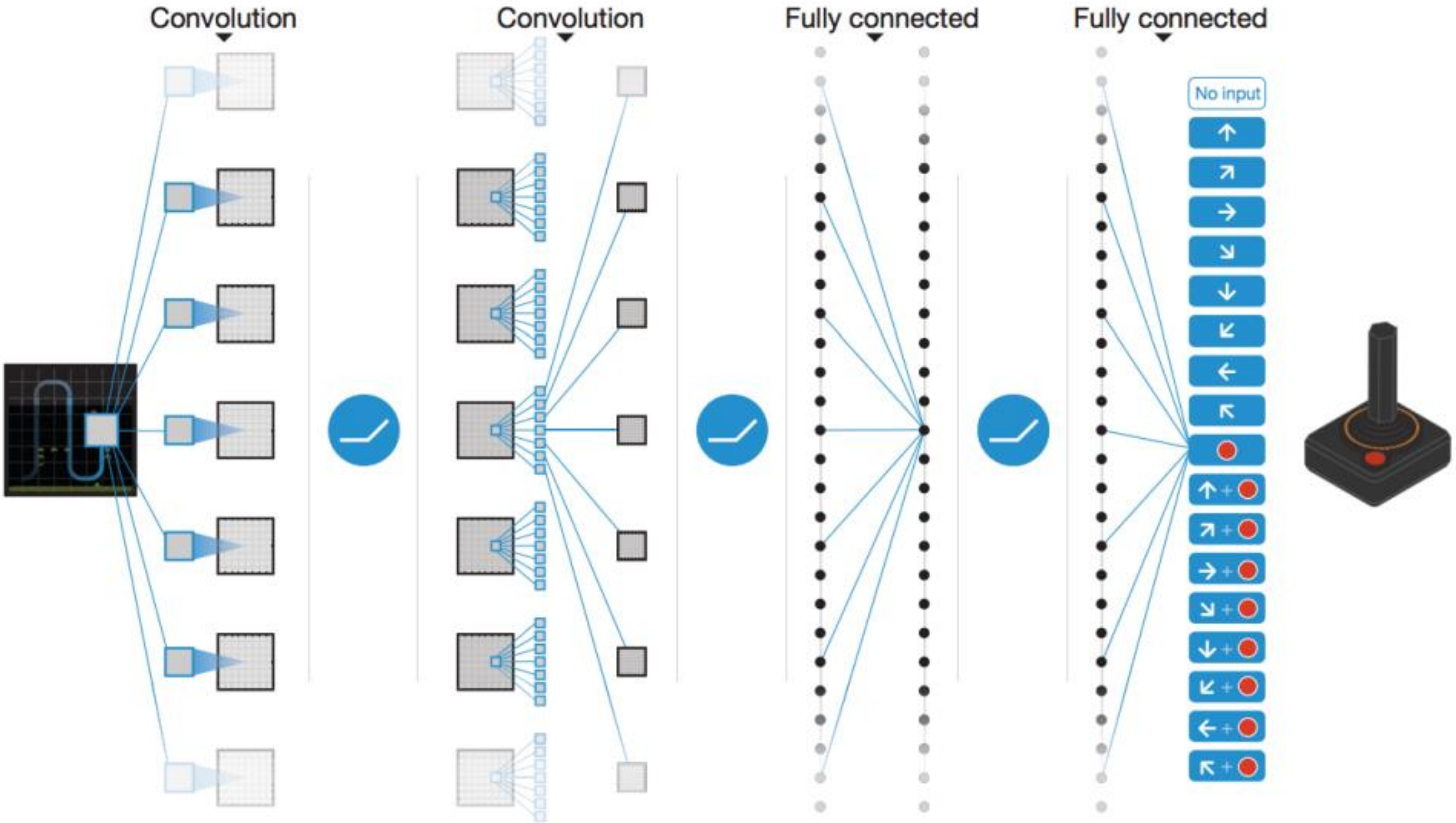
Update the behavior network

DQN in Atari

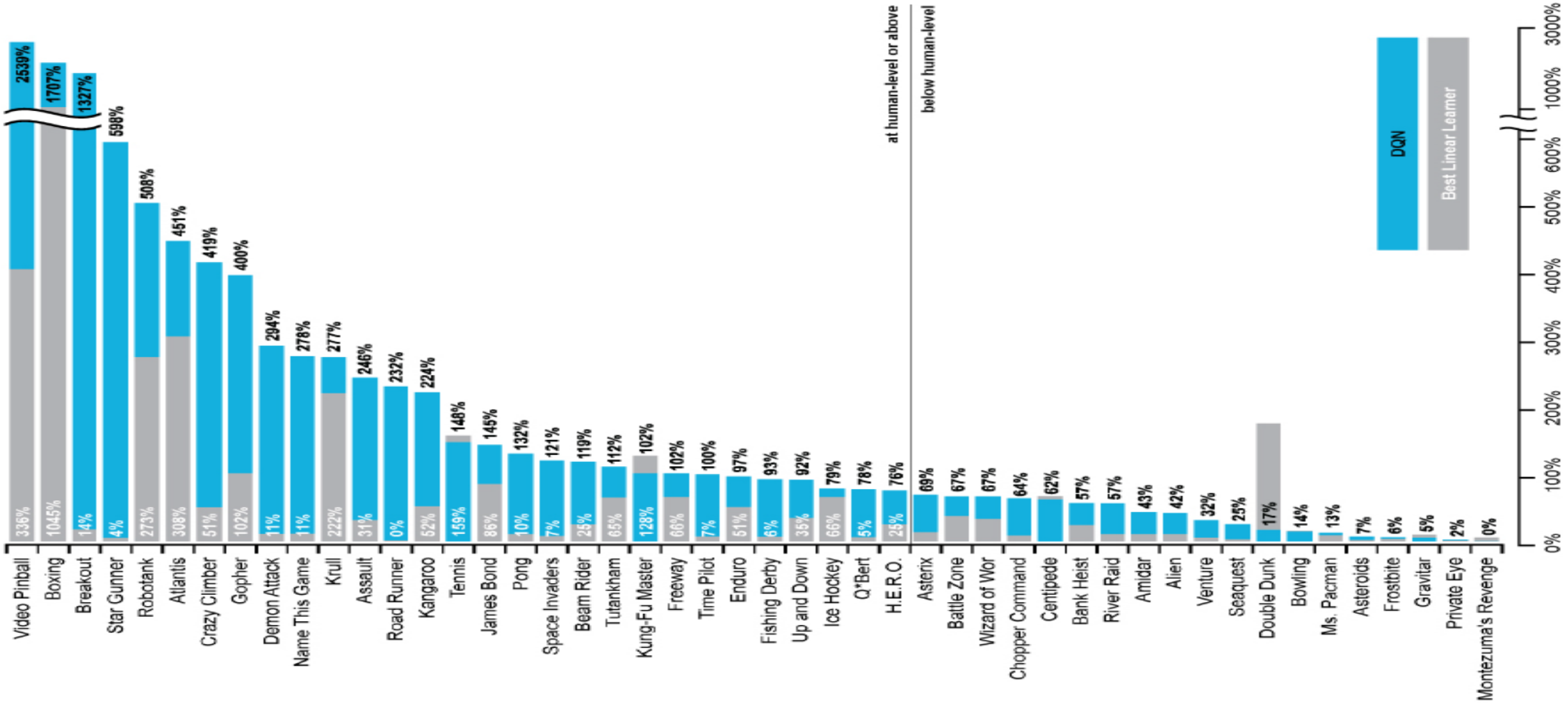
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step


32 4x4 filters
256 hidden units
Fully-connected linear output layer
16 8x8 filters
4x84x84
Stack of 4 previous frames
Convolutional layer of rectified linear units
Convolutional layer of rectified linear units
Fully-connected layer of rectified linear units

Network architecture and hyperparameters fixed across all games


Convolution
Convolution
Fully connected
Fully connected
No input

DQN Results in Atari

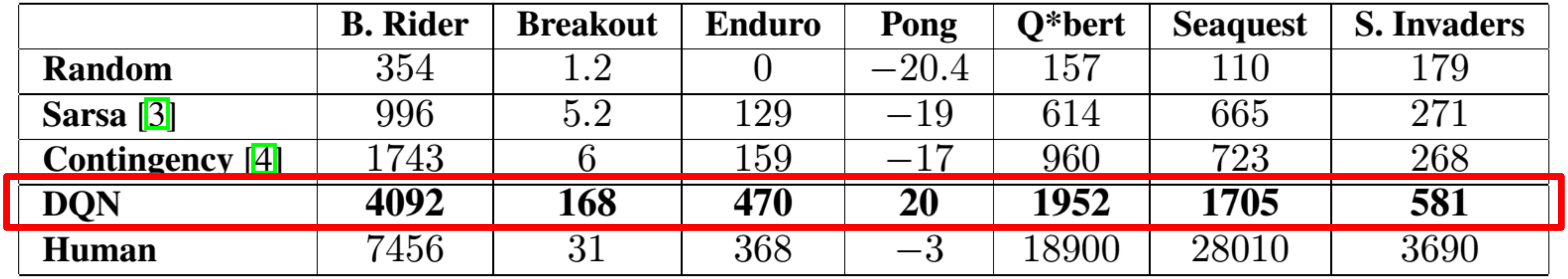


I-Chen Wu

How much does DQN help?

	Q-learning	Q-learning +Target Q	Q-learning + Replay	Q-learning + Replay +Target Q
Breakout	3	10	241	317
Enduro	29	142	831	1006
River	1453	2868	4103	7447
Seaquest	276	1003	823	2894
Space Invaders	302	373	826	1089

Experiments - DQN



	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690

- The upper table compares average total reward for various learning methods by running an ε-greedy policy with ε = 0.05 for a fixed number of steps.

Value-Based Reinforcement Learning

- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

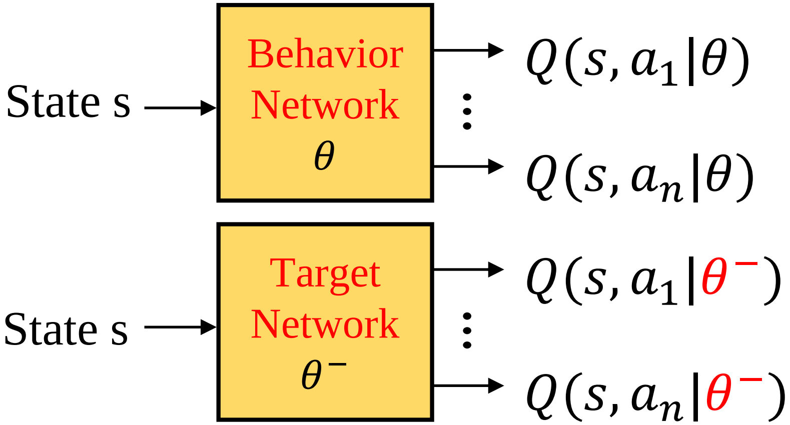
Double DQN (DDQN)

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

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

⇩

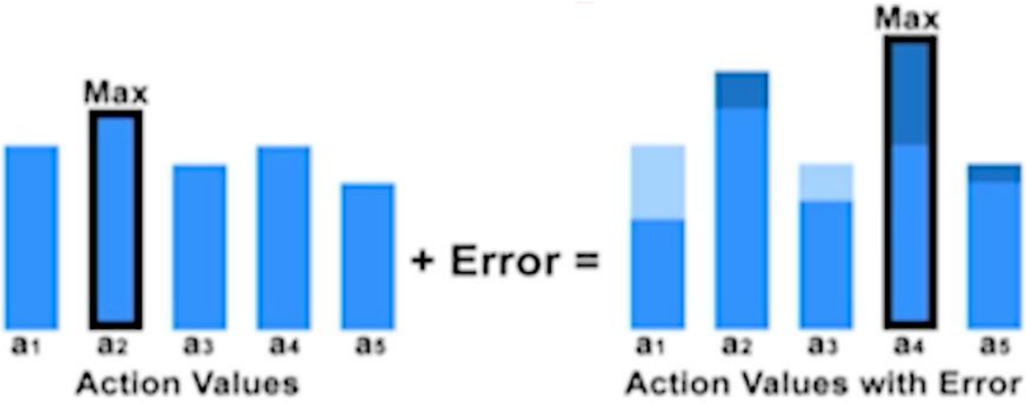
$$Y_t^{DoubleQ} = r_{t+1} + \gamma Q\left(S_{t+1}, \boxed{\underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a|\theta)} | \theta^{-}\right)$$

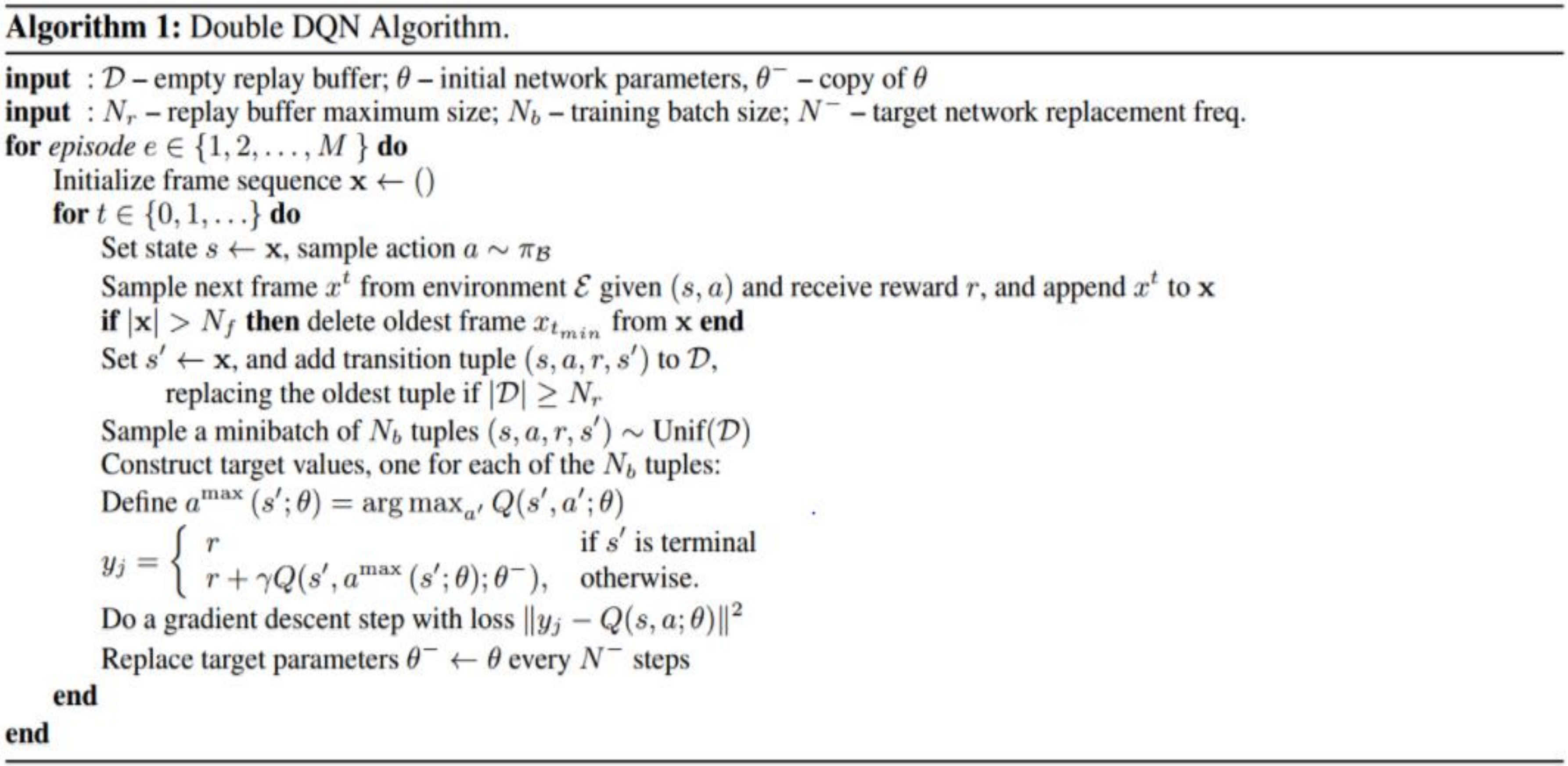


Overestimation Problem

- Q-Learning update

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$



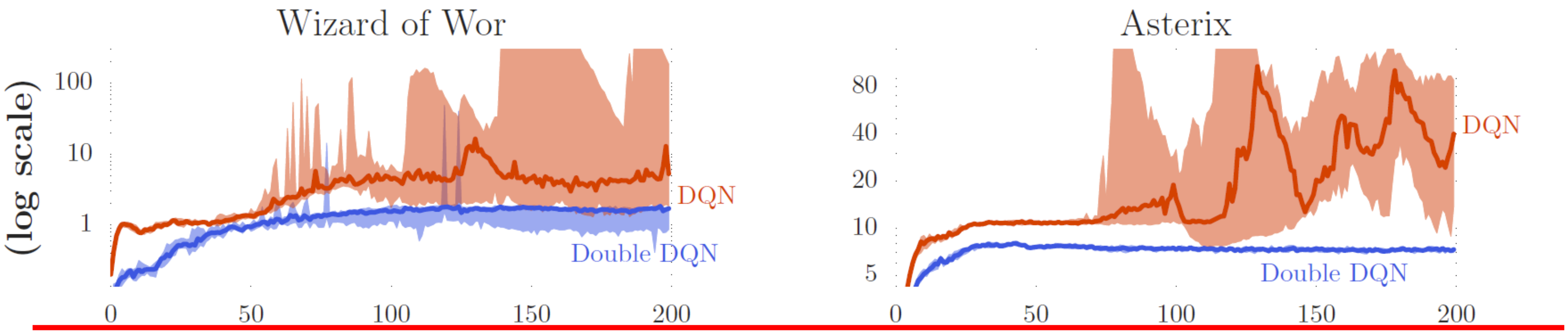


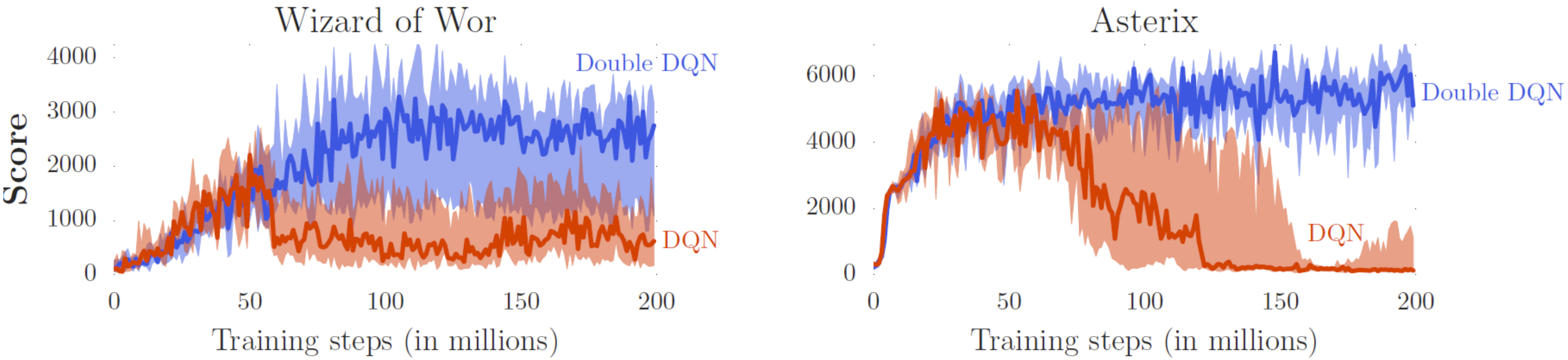
Algorithm 1: Double DQN Algorithm.

```
input : $\mathcal{D}$ – empty replay buffer; $\theta$ – initial network parameters, $\theta^-$ – copy of $\theta$
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
```

Experiments - DDQN

Predicted Q-value at training (showing over-optimism)





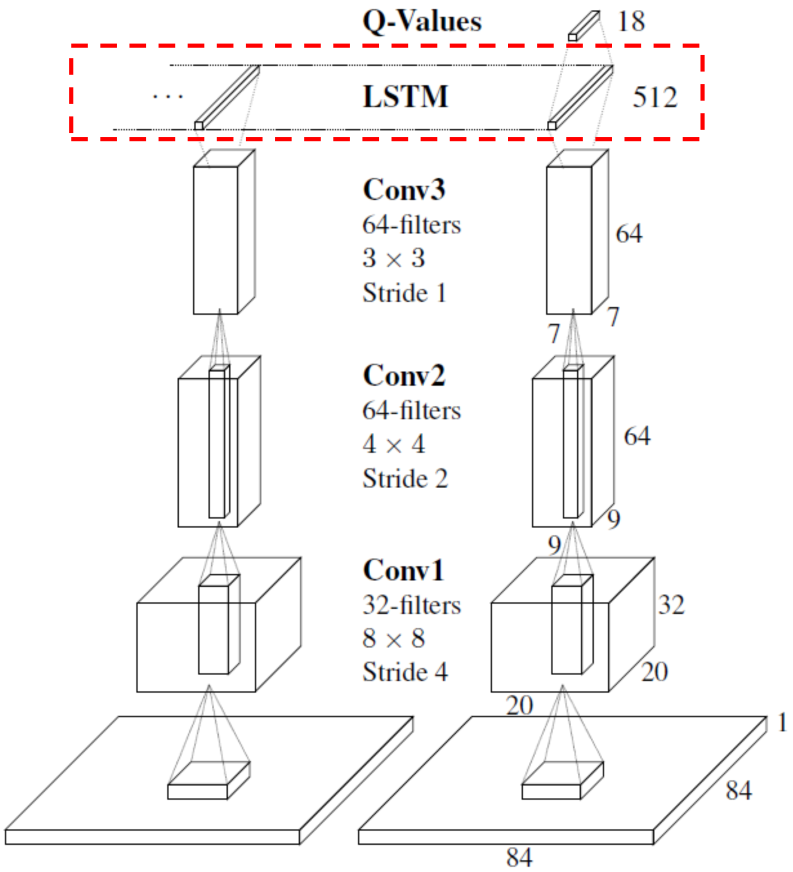
Real Scores

Value-Based Reinforcement Learning

- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

Deep Recurrent Q-Network (DRQN)

- Replace only its first fully-connected layer with a LSTM.
- Take a single 84 x 84 preprocessed image (not 4 consecutive images)
- Finally, LSTM outputs become Q-values for each action after passing through a fully-connected layer.



Update for DRQN

Update: episodes are selected randomly from the replay memory (for example, choose an episode with $s_1, \ldots, s_{200}$ states)

- Sequential:
  - Updates begin at the beginning of the episode to the end of the episode (always start from s_1)
  - Good for LSTM, but violate DQN’s random sampling policy
- Random:
  - Updates begin at random points in the episode and proceed for only *unroll iterations* time-steps (randomly pick s_i)
  - LSTM’s hidden state must be zeroed at the start of each update. Harder for the LSTM to learn.
- Both have similar performance (use random here)

DRQN Results

- It can generalize its policies to the case of complete observations (on Flickering Pong)
- DRQN's performance generalizes better than DQN's at all levels of partial information

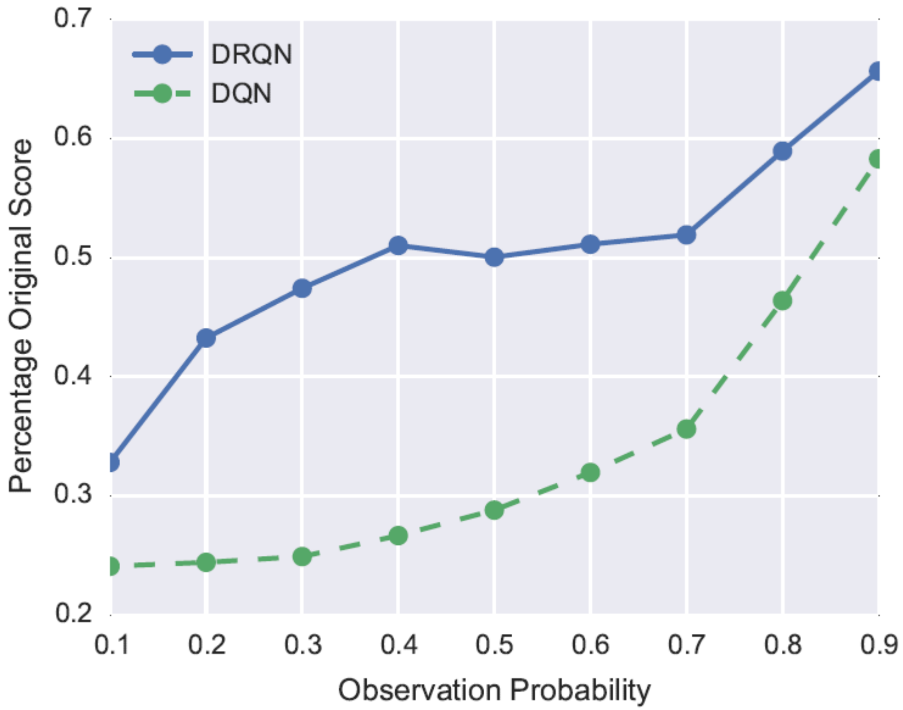


Figure 5: When trained on normal games (MDPs) and then evaluated on flickering games (POMDPs), DRQN's performance degrades more gracefully than DQN's.

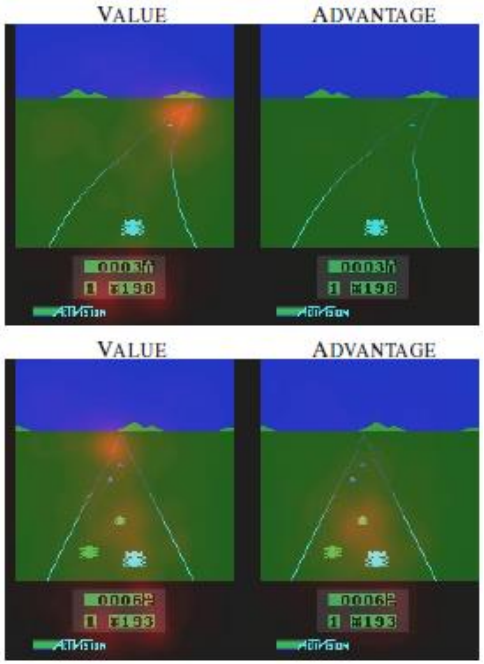
I-Chen Wu

Value-Based Reinforcement Learning

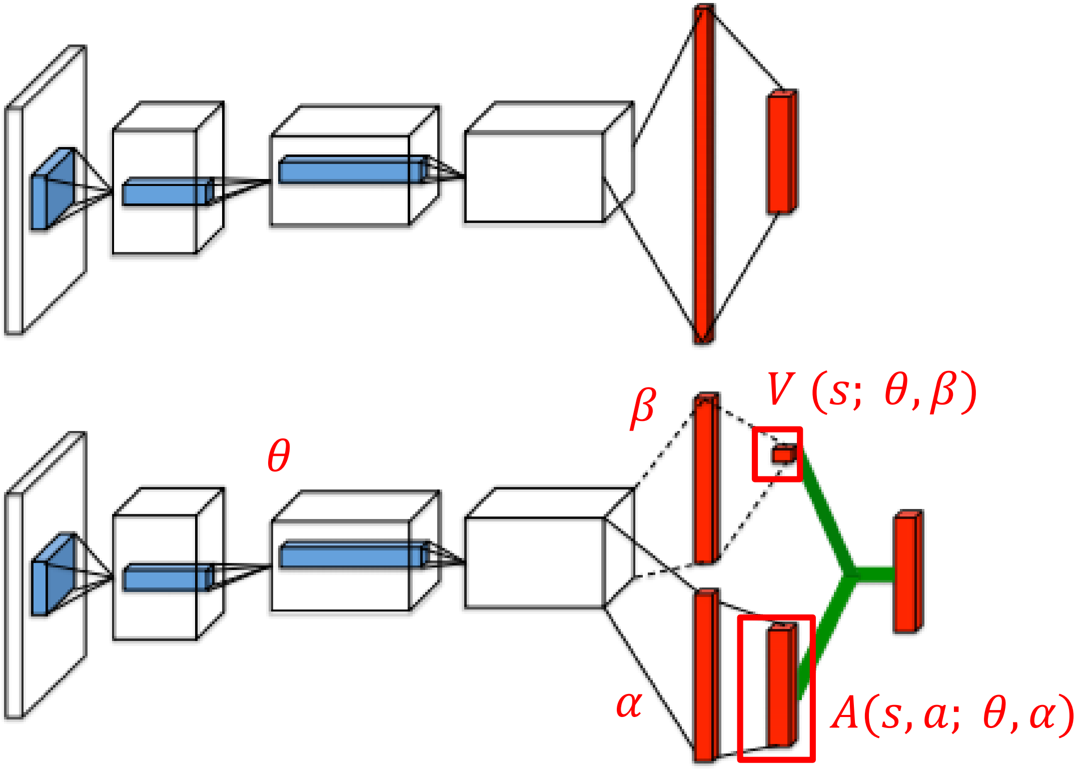
- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

Dueling Network

- In most states, learning the effect of each action is not necessary.
  - Actions do not affect the environment in any relevant way
- Intuitively, the dueling architecture can learn whether states are valuable (or not).
  - Advantage stream
  - Value stream



Dueling Network



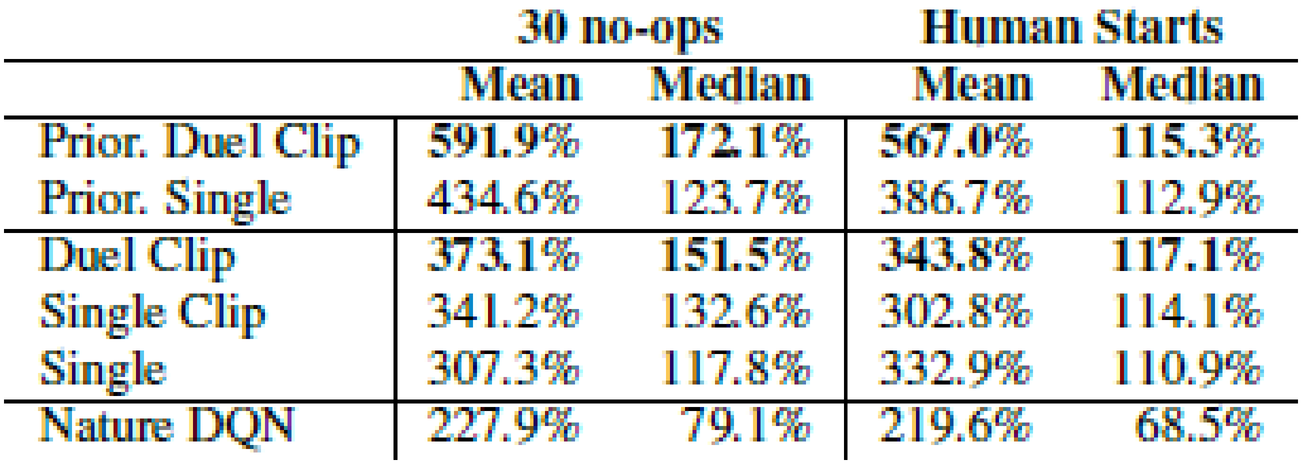
Q-network (top) and the dueling Q-network (bottom). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module combines them by the equation $Q(s,a;\ \theta,\alpha,\beta) = V(s;\ \theta,\beta) + A(s,a;\ \theta,\alpha)$. Both networks output Q-values for each action.

Dueling Network

- A relative measure of the importance of each action
  - $Q(s,a;\ \theta,\alpha,\beta) = V\ (s;\ \theta,\beta)\ + A(s,a;\ \theta,\alpha)$
    - Unidentifiable in the sense that given Q we cannot recover V and A uniquely.
- Address the issue of identifiability
  - $Q(s,a;\ \theta,\alpha,\beta) =\ V\ (s;\ \theta,\beta)\ + (A(s,a;\ \theta,\alpha) - \max\limits_{a'\in|A|} A(s,a';\ \theta,\alpha))$
    - Force the advantage function estimator have zero advantage at the chosen action
    - When $a^* = \max\limits_{a'} Q(s,a')$, $Q(s,a^*) = V(s)$
- Improvement (increase stability)
  - $Q(s,a;\ \theta,\alpha,\beta)\ =\ V\ (s;\ \theta,\beta)\ + (A(s,a;\ \theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a'\ ;\ \theta,\alpha))$
    - When $a^* = \max\limits_{a'} Q(s,a')$, $Q(s,a^*) \neq V(s)$
    - The advantages only need to change as fast as the mean.

Experiments – Dueling Network

- Achieve human level performance on **42 out of 57** games



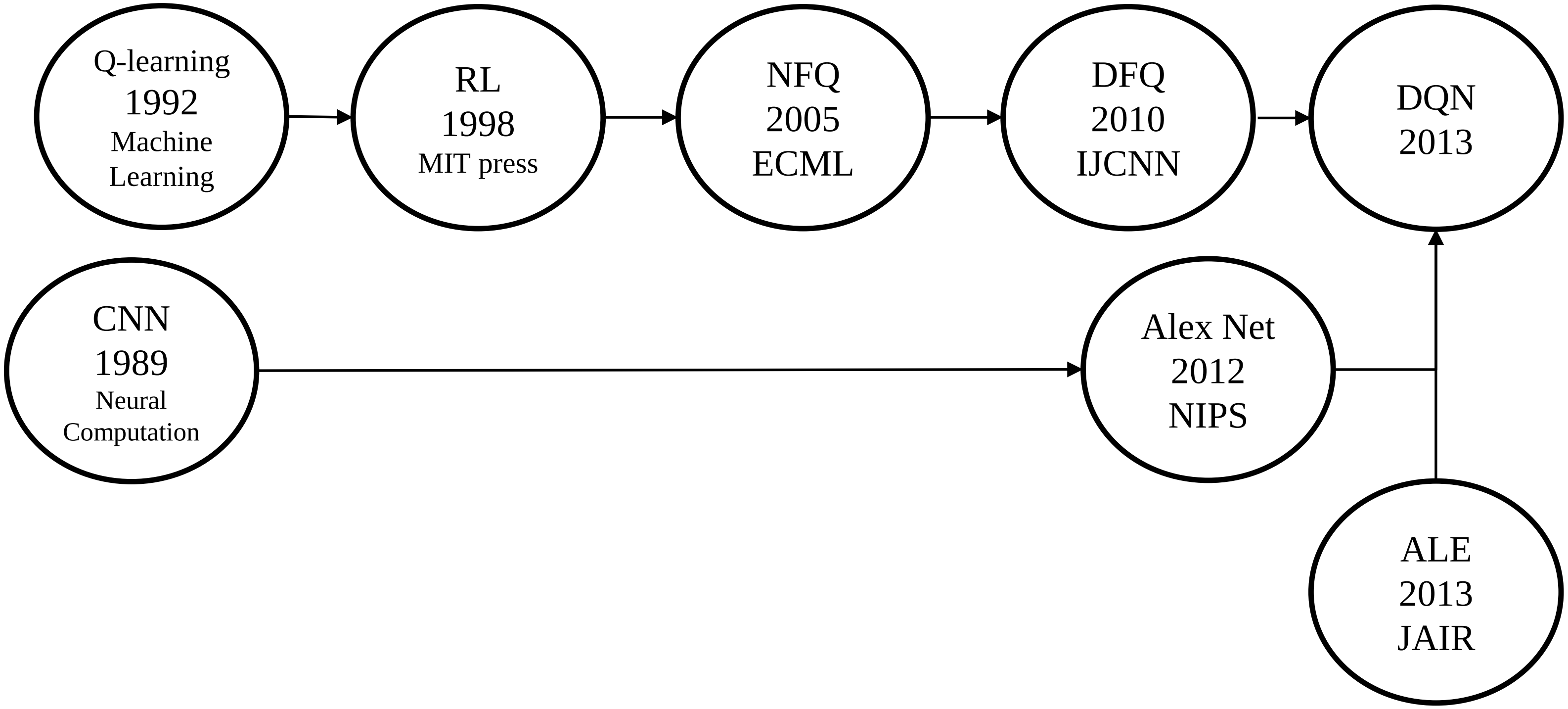
	30 no-ops		Human Starts	
	Mean	Median	Mean	Median
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%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

Measured in percentage of human performance

Value-Based Reinforcement Learning

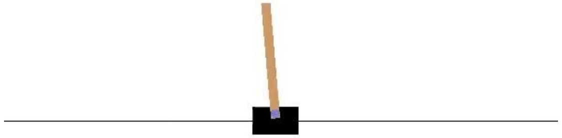
- Fundamentals
  - Model Free Reinforcement Learning
  - ε-Greedy Exploration, Q-Learning
  - Function Approximation
- Algorithms
  - DQN, DDQN (Double DQN), DRQN
  - Dueling Network (with Advantage)
  - Others

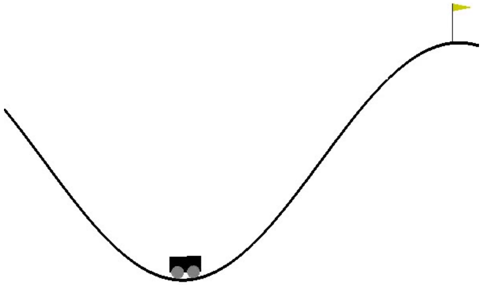
DQN – Genealogy



Before DQN

- NFQ (Neural Fitted Q Iteration, 2005)
  - *First Experiences with a Data Efficient Neural Reinforcement Learning Method*
  - Using neural network (MLP)
    - 2 hidden layers with 5 neurons
  - Using experience replay
    - collected in triples of the form (s,a,s′)
  - Internal state:
- DFQ (Deep Fitted Q Iteration, 2010)
  - Applying deep learning (MLP, but not CNN)
    - Deep auto-encoders
      - 21 layers
        - 900-900-484-225-121-113-57-29-15-8-2-8-15-29-57-113-121-225-484-900-900 neurons
  - AutoEncoder visualization (2D latent space) to train policy.





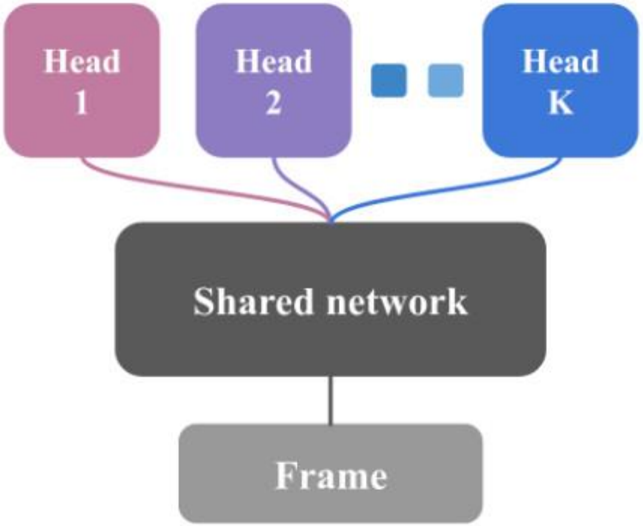
DQN – Summary

- Using convolutional neural network (CNN)
  - Alex Net (2012)
    - ReLU
    - GPU

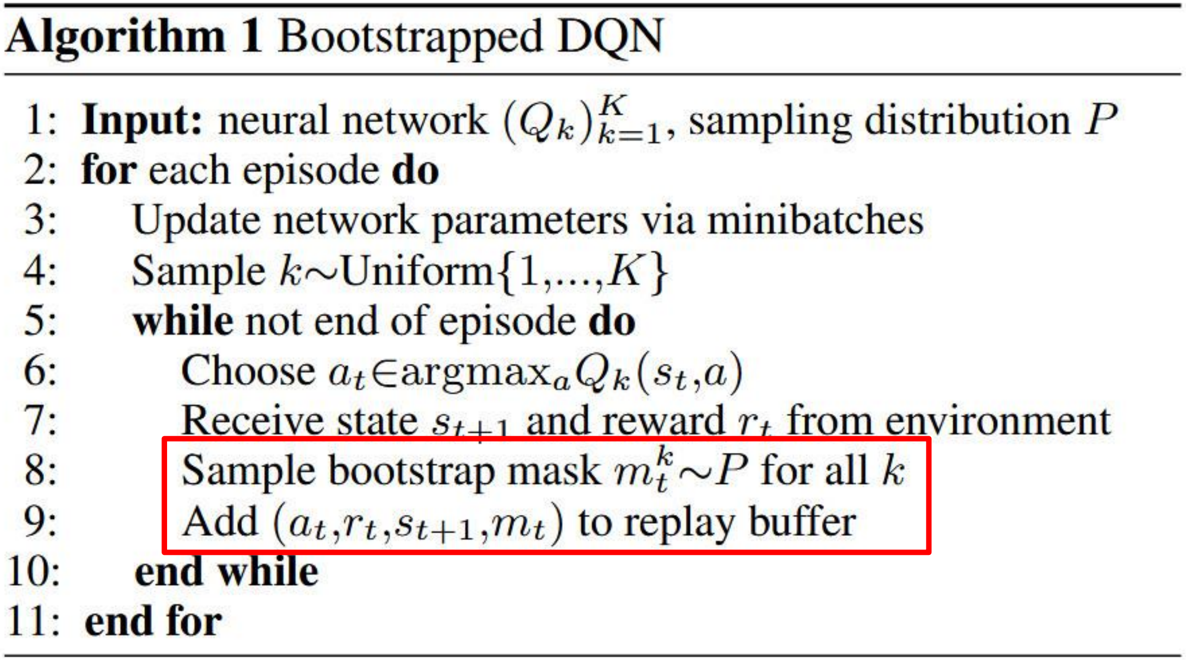
- Arcade Learning Environment (ALE), 2013
  - For Atari games
  - An Evaluation Platform for General Agents
- Using experience replay

Bootstrapped DQN – Summary

- Bootstrapped with k-heads DQN
- No ϵ-greedy
  - ϵ select a head at episode initial
    - Greedy with this head
  - Deep exploration
- * A way go to distribution
  - Bootstrapped distribution



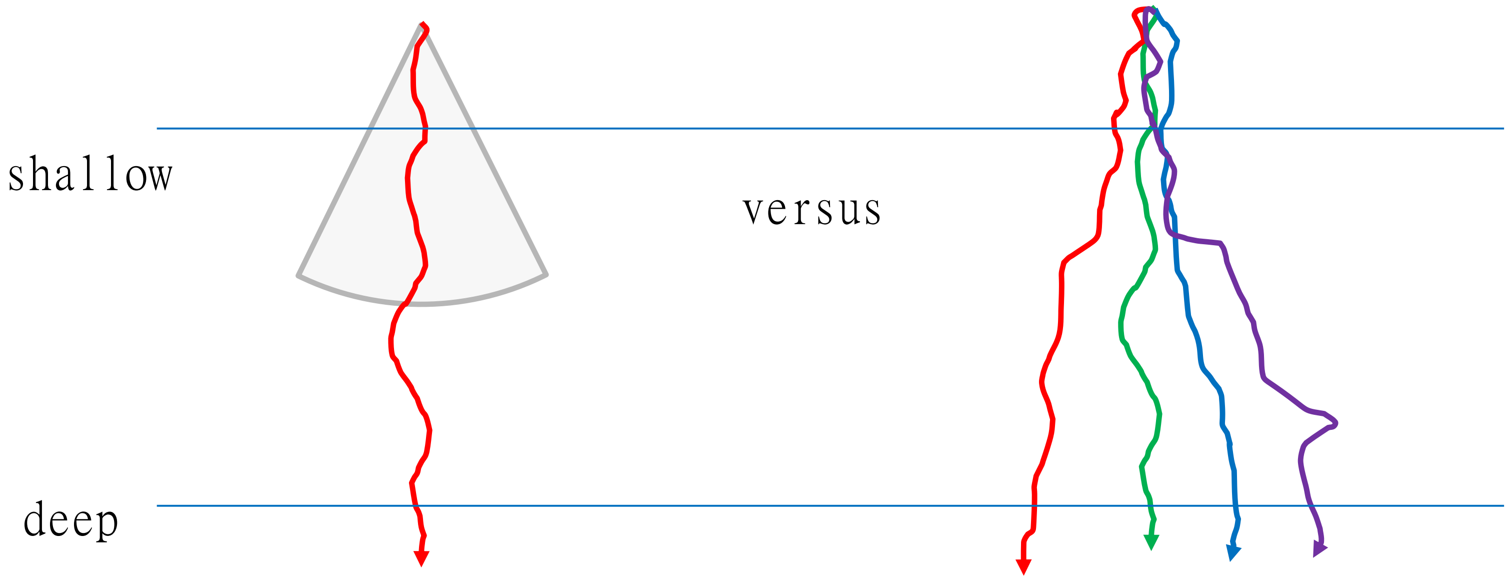
Bootstrapped DQN – Method



Algorithm 1 Bootstrapped DQN

```
**Input:** neural network $(Q_k)_{k=1}^{K}$, sampling distribution $P$
**for** each episode **do**
    Update network parameters via minibatches
    Sample $k \sim \text{Uniform}\{1,\ldots,K\}$
    **while** not end of episode **do**
        Choose $a_t \in \text{argmax}_a Q_k(s_t,a)$
        Receive state $s_{t+1}$ and reward $r_t$ from environment
        Sample bootstrap mask $m_t^k \sim P$ for all $k$
        Add $(a_t,r_t,s_{t+1},m_t)$ to replay buffer
    **end while**
**end for**
```

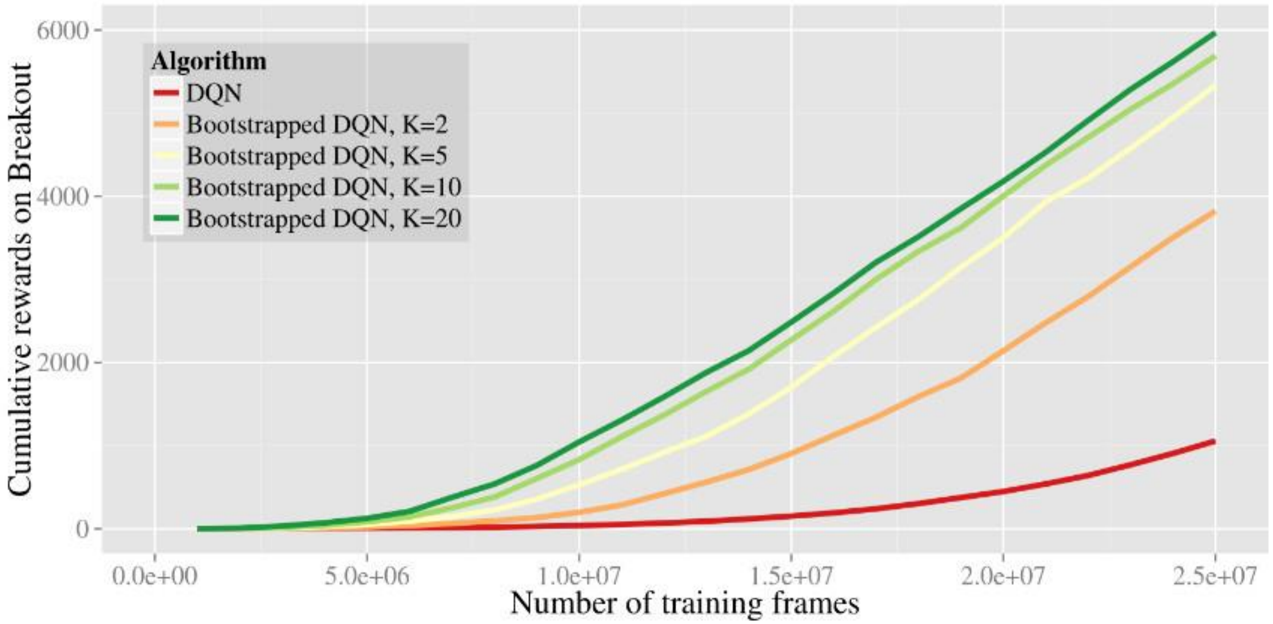
Bootstrapped DQN – Deep Exploration



▨: ε-greedy explored states
🟥🟩🟦🟪: different greedy paths

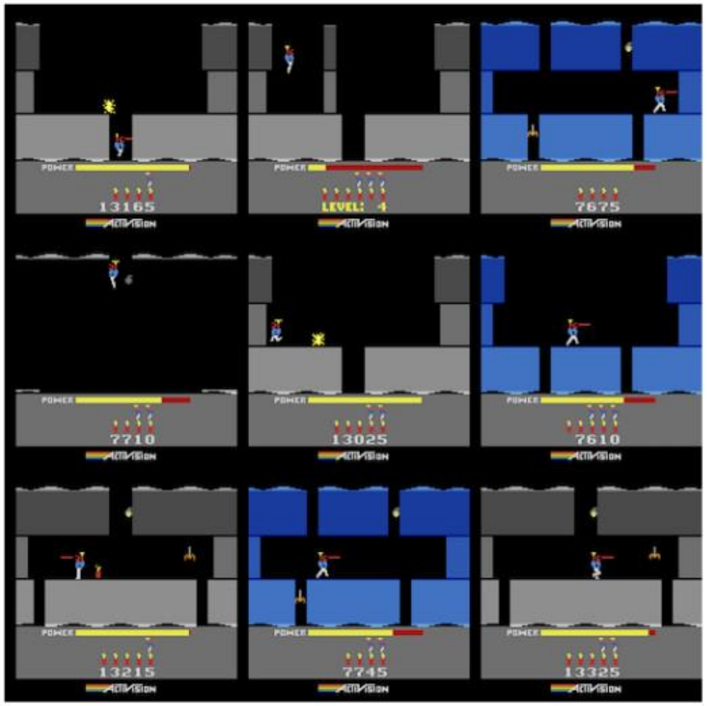
Bootstrapped DQN – Result

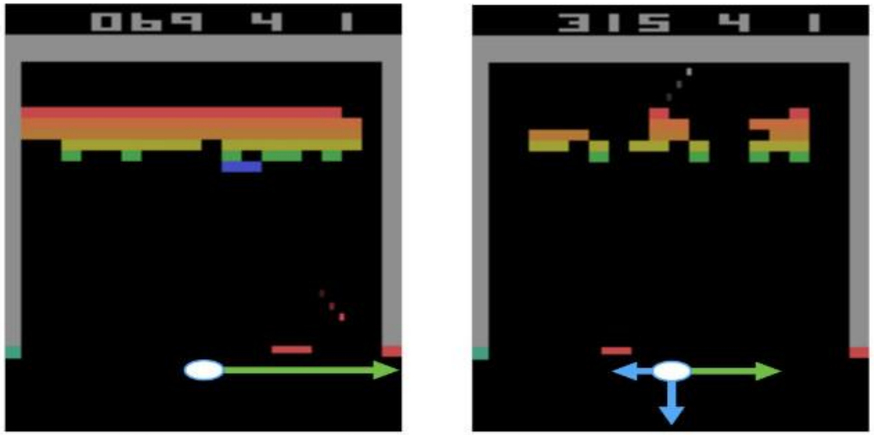
- Compare to DQN (Different head count, in Breakout)



Bootstrapped DQN – Diverse exploration policies

- Compare to DQN (faster, stronger)

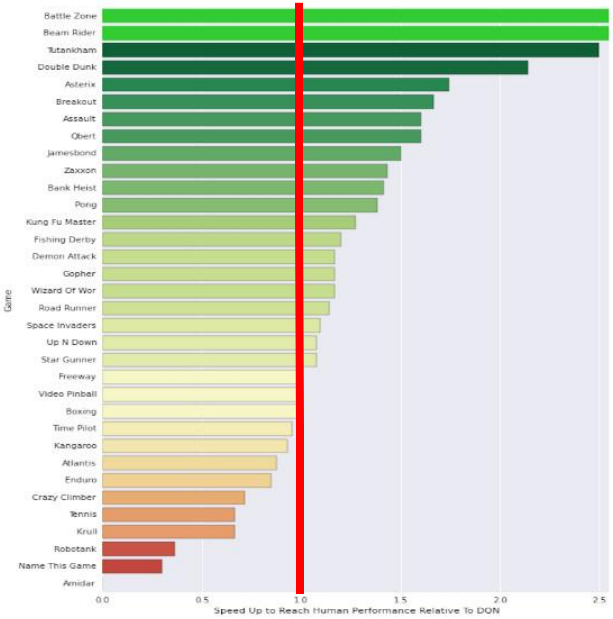




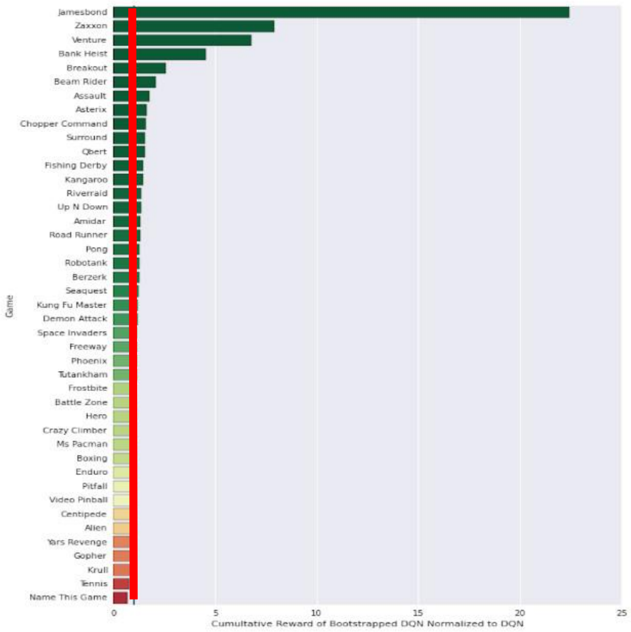
(a) All heads vote right. (b) Heads disagree on policy.

Bootstrapped DQN - Result

- Compare to DQN (faster, stronger)



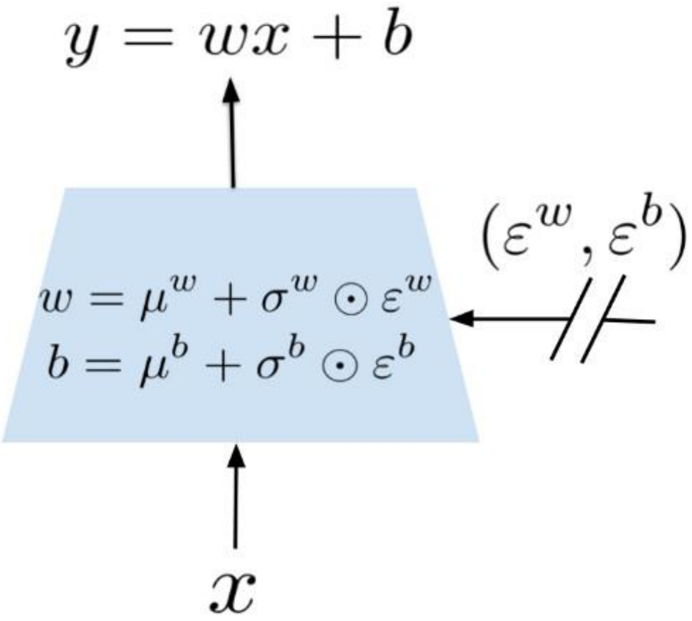
Bootstrapped DQN at human level faster than DQN.



Bootstrapped DQN improves cumulative rewards.

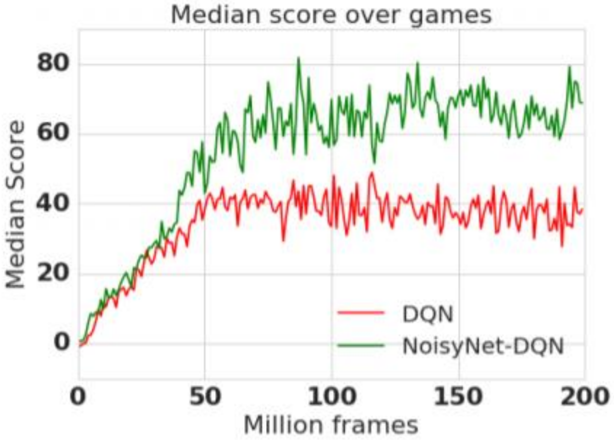
NoisyNet – Summary

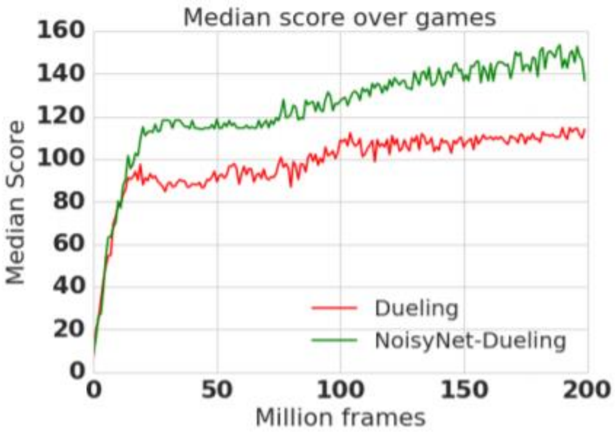
- Add trainable noise to neural network
  - Stochastic policy (through latent state space)
    - No ϵ-greedy
  - Better optimization
- A way go to distribution
  - [ICLR 2018]

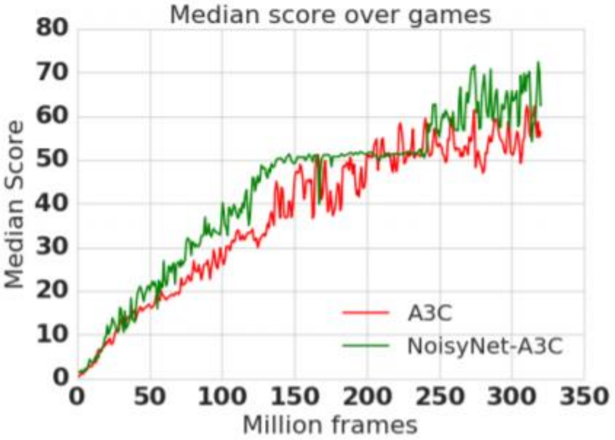

$y = wx + b$
$w = \mu^w + \sigma^w \odot \varepsilon^w$
$b = \mu^b + \sigma^b \odot \varepsilon^b$
$(\varepsilon^w, \varepsilon^b)$
x

NoisyNet – Result

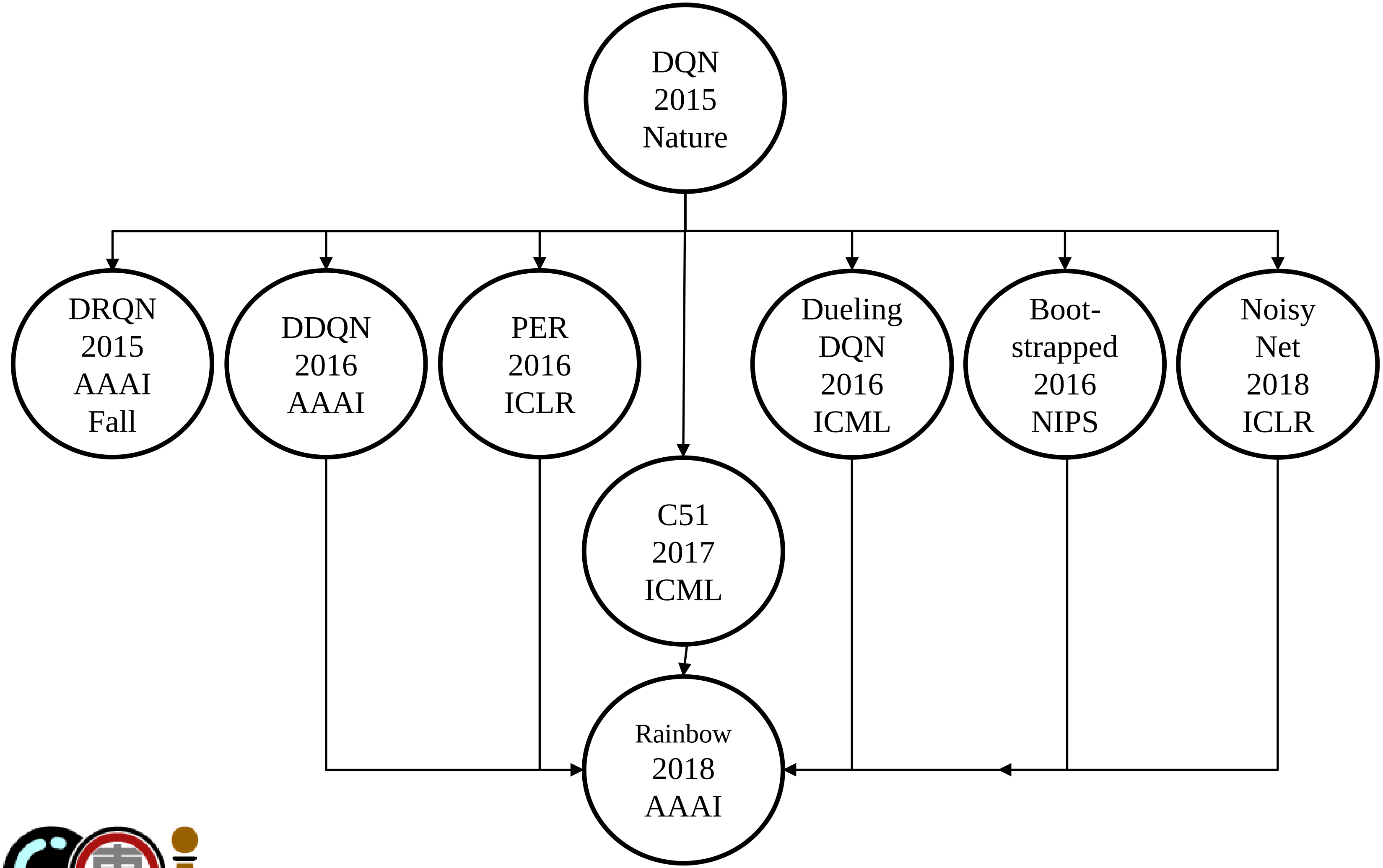
- Better than DQN & Dueling (using e-greedy)
- But, close to A3C.



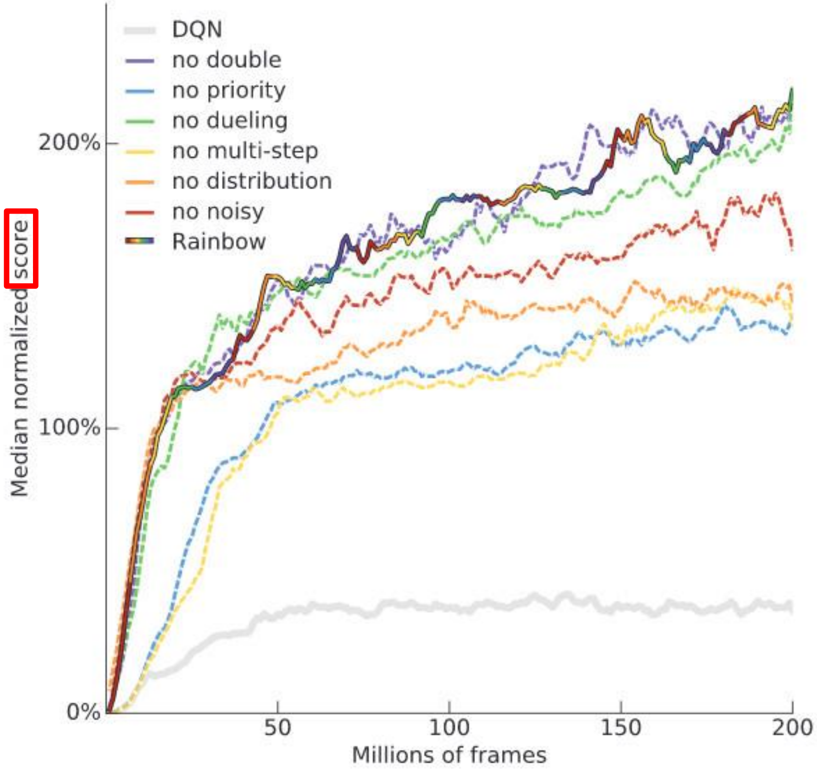




Rainbow - Genealogy



Rainbow – Ablation studies



- Prioritized replay and multi-step
  - the two most crucial components
- Distributional Q-learning
  - Perform after 40 million frames
  - Relatively to human performance
- Noisy Nets
  - ϵ- greedy when removed
  - large drop in performance for several games
- Dueling network
  - median score/above-human performance levels may hide the impact
- Double Q-learning
  - Actual returns are often higher than10
    - Underestimated
  - May increase if the support of the distributions is expanded