Winter School 2024
Reinforcement Learning

Advanced Reinforcement Learning

Dr. Lorenzo Brigato

Artificial Intelligence in Health and Nutrition (AIHN) Laboratory
ARTORG Center for Biomedical Engineering Research
University of Bern

Outlook

- Value Function Approximation
- Policy Gradient
- Deep Reinforcement Learning

Large-Scale Reinforcement Learning

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Go: 10^{170}
  - Robots: continuous state space
- How can we scale up e.g., the model-free methods for prediction and control seen before?

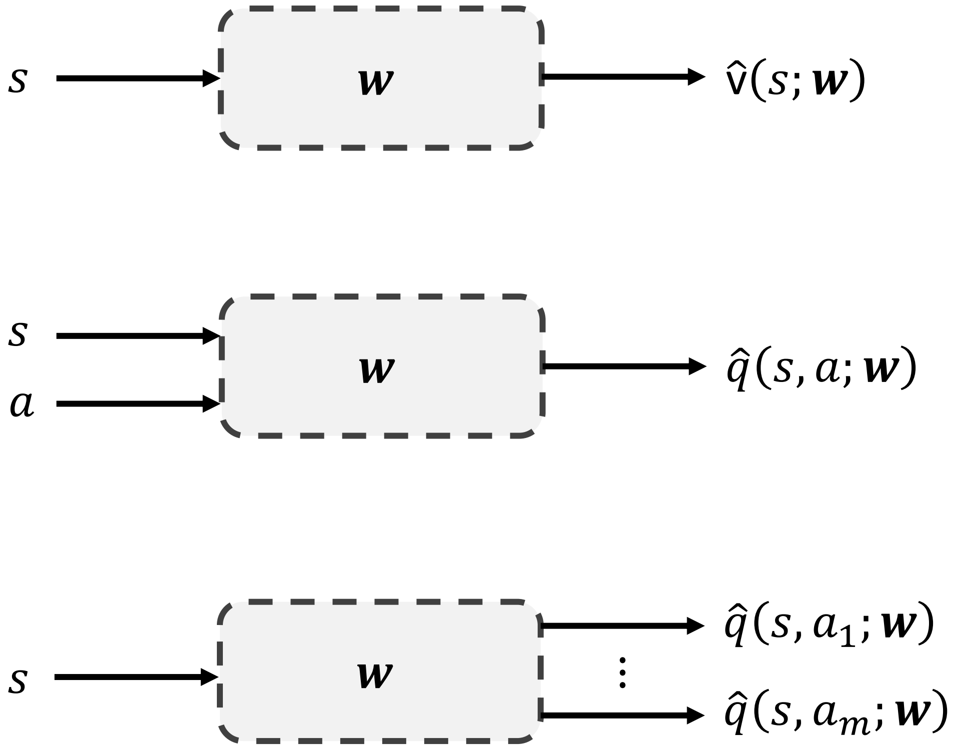
Value Function Approximation

- 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

$$\hat{\mathrm{v}}(s; \boldsymbol{w}) \approx \mathrm{v}_\pi(s)$$
$$\hat{q}(s, a; \boldsymbol{w}) \approx q_\pi(s, a)$$

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

Types of Value Function Approximation



Value Function Approx. By Stochastic Gradient Descent

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

$$J(\boldsymbol{w}) = \mathbb{E}_\pi[(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s;\boldsymbol{w}))^2]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta\boldsymbol{w} &= -\frac{1}{2}a\nabla_w J(\boldsymbol{w})\\ &= \alpha\mathbb{E}_\pi[(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s;\boldsymbol{w}))\nabla_w\hat{\mathrm{v}}(s;\boldsymbol{w})]\end{aligned}$$

- Stochastic gradient descent samples the gradient

$$\Delta\boldsymbol{w} = \alpha(\mathrm{v}_\pi(s) - \hat{\mathrm{v}}(s;\boldsymbol{w}))\nabla_w\hat{\mathrm{v}}(s;\boldsymbol{w})$$

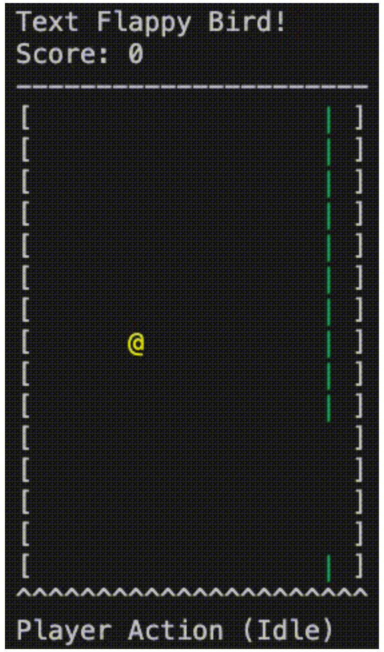
- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a feature vector

$$x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

- E.g., Polynomials, Fourier Basis
- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess



```
Text Flappy Bird!
Score: 0
----------------------
[                  | ]
[                  | ]
[                  | ]
[                  | ]
[                  | ]
[                  | ]
[                  | ]
[      @           | ]
[                  | ]
[                  | ]
[                    ]
[                    ]
[                    ]
[                    ]
[                  | ]
^^^^^^^^^^^^^^^^^^^^^^
Player Action (Idle)
```

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(s;\boldsymbol{w}) = x(s)^T\boldsymbol{w} = \sum_{i=0}^{n} x_i(s)w_i$$

- Objective function is quadratic in parameters $\boldsymbol{w}$

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

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

$$\nabla_w \hat{v}(s;\boldsymbol{w}) = x(s)$$
$$\Delta\boldsymbol{w} = \alpha\big(v_\pi(s) - \hat{v}(s;\boldsymbol{w})\big)x(s)$$

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

Incremental Prediction Algorithms

- Assumed true value $v_{\pi}(s)$ function 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

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

  - For TD(0), the target is the TD target $G_{t:t+1} = R_{t+1} + \gamma\, \hat{v}(s_{t+1}; \boldsymbol{w})$

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

Monte-Carlo with Value Function Approximation (1/2)

- Return G_t is an unbiased, noisy sample of true value $\mathrm{v}_\pi(s_t)$
- Can therefore apply supervised learning to "training data":

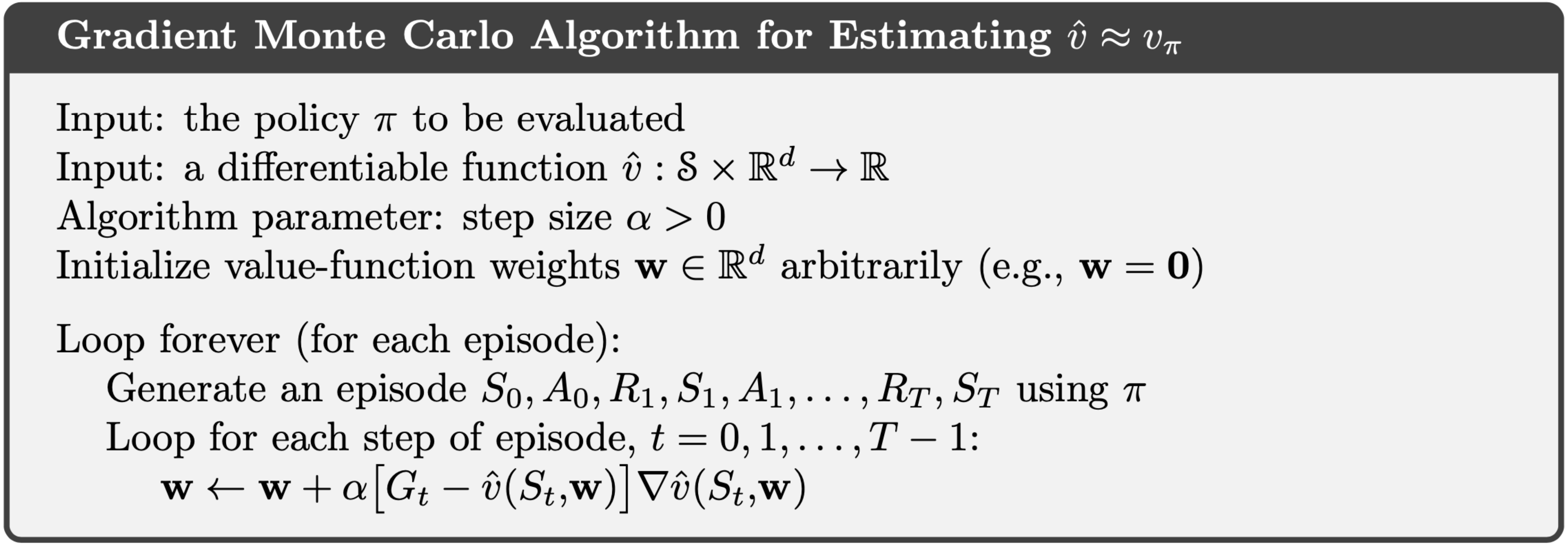
$$< S_1, G_1 >, < S_2, G_2 >, \dots, < S_T, G_T >$$

- For example, using linear Monte-Carlo policy evaluation

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha\big(\boldsymbol{G_t} - \hat{\mathrm{v}}(S_t; \boldsymbol{w})\big)\nabla_w \hat{\mathrm{v}}(S_t; \boldsymbol{w}) \\ &= \alpha\big(G_t - \hat{\mathrm{v}}(S_t; \boldsymbol{w})\big)\boldsymbol{x}(S_t)\end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

Monte-Carlo with Value Function Approximation (2/2)



Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using $\pi$
    Loop for each step of episode, $t = 0, 1, \dots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$
```

[An Introduction to Reinforcement Learning, Sutton and Barto]

TD Learning with Value Function Approximation (1/2)

- The TD-target $G_{t:t+1} = R_{t+1} + \gamma\, \hat{v}(s_{t+1}; \boldsymbol{w})$ is a biased sample of true value $v_\pi(s_t)$
- Can still apply supervised learning to "training data":

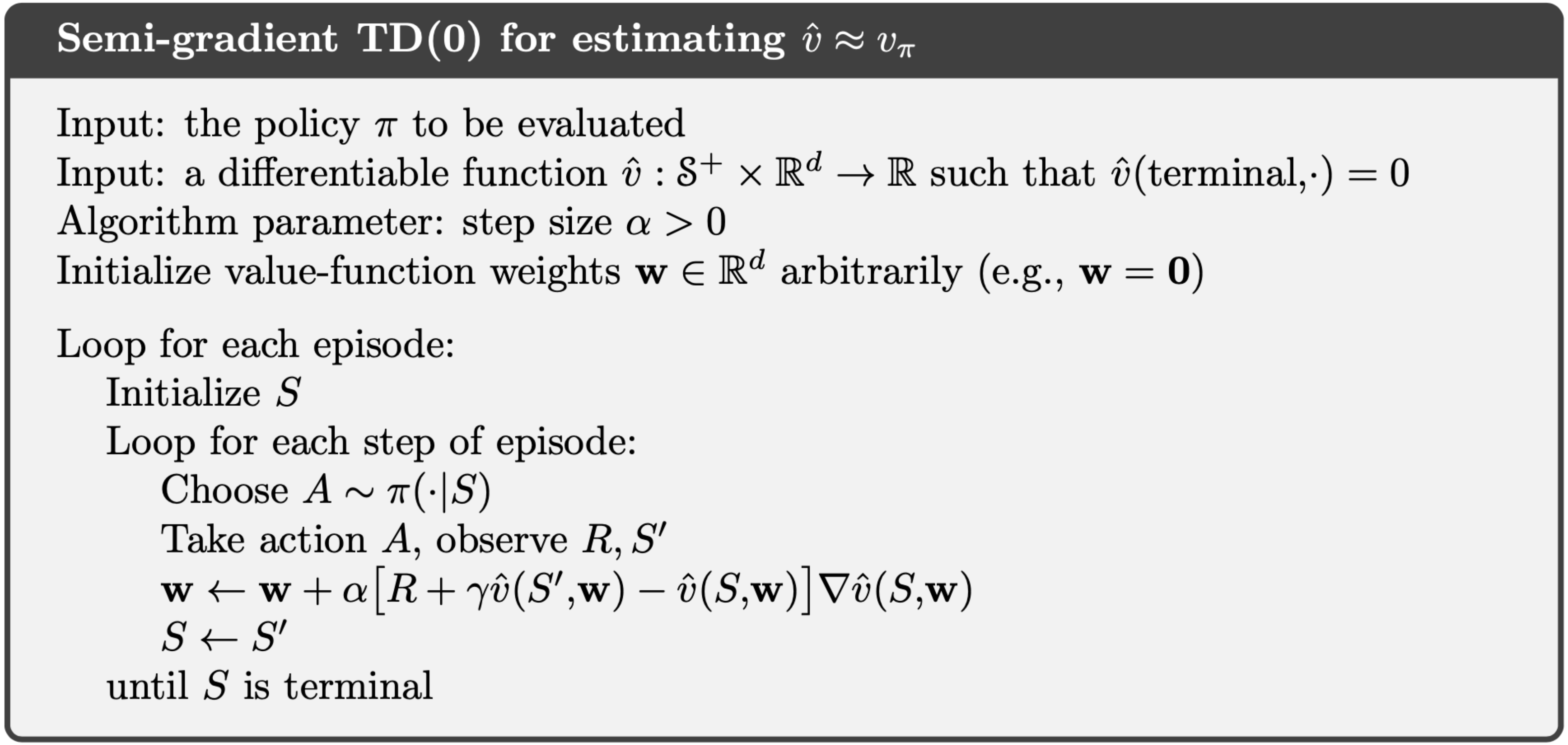
$$< S_1, R_1 + \gamma\, \hat{v}(S_2; \boldsymbol{w}) >, < S_2, R_3 + \gamma\, \hat{v}(S_3; \boldsymbol{w}) >, \dots, < S_{T-1}, R_T >$$

- For example, using linear TD(0)

$$\begin{aligned} \Delta \boldsymbol{w} &= \alpha\big(R + \gamma\, \hat{v}(S'; \boldsymbol{w}) - \hat{v}(S_t; \boldsymbol{w})\big)\nabla_w \hat{v}(S_t; \boldsymbol{w}) \\ &= \alpha \delta \boldsymbol{x}(S) \end{aligned}$$

- (Semi-Gradient) We do not consider the effect of changing $\boldsymbol{w}$ on the target
- Linear TD(0) converges (close) to global optimum

TD Learning with Value Function Approximation (2/2)



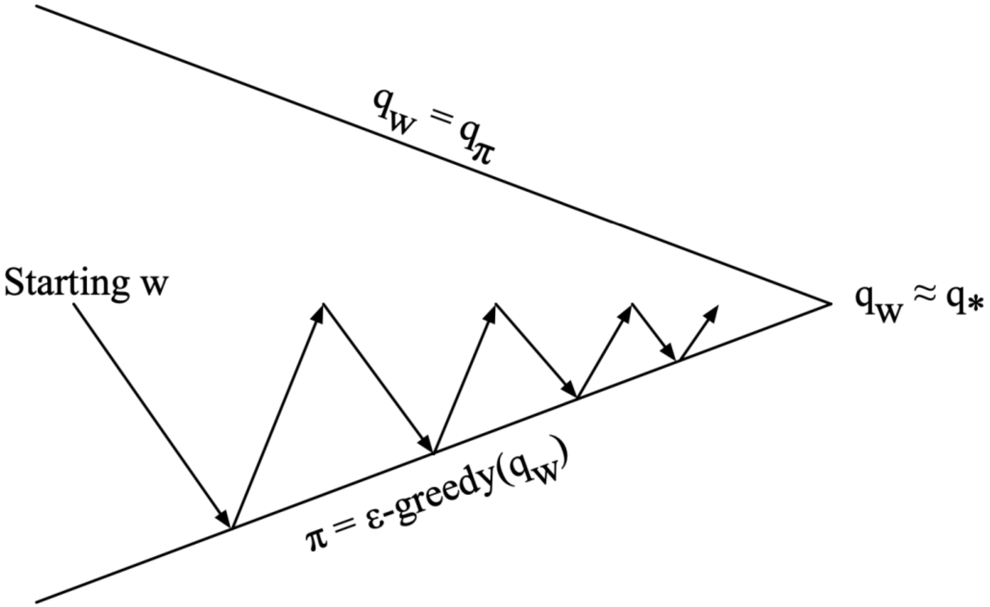
Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R, S'$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until $S$ is terminal
```

[An Introduction to Reinforcement Learning, Sutton and Barto]

Control with Value Function Approximation



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

[David Silver, IRL, UCL 2015]

Action-Value Function Approximation

- Approximate the action-value function

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

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

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

- Use stochastic gradient descent to find a local minimum

$$\begin{aligned} -\frac{1}{2}\nabla_w J(\boldsymbol{w}) &= \left(q_\pi(S, A) - \hat{q}(S, A; \boldsymbol{w})\right)\nabla_w \hat{q}(S, A; \boldsymbol{w}) \\ \Delta\mathbf{w} &= \alpha\left(q_\pi(S, A) - \hat{q}(S, A; \boldsymbol{w})\right)\nabla_w \hat{q}(S, A; \boldsymbol{w}) \end{aligned}$$

Linear Action-Value Function Approximation

- Represent state and action by a feature vector

$$x(S, A) = \begin{pmatrix} x_1(S,A) \\ \vdots \\ x_n(S,A) \end{pmatrix}$$

- Represent action-value function by linear combination of features

$$\hat{q}(S, A, \boldsymbol{w}) = x(S, A)^T \boldsymbol{w} = \sum_{i=0}^{n} x_i(S, A) w_i$$

- Stochastic gradient descent update

$$\nabla_w \hat{q}(S, A; \boldsymbol{w}) = x(S, A)$$
$$\Delta \mathbf{w} = \alpha\big(q_\pi(S, A) - \hat{q}(S, A; \boldsymbol{w})\big) x(S, A)$$

Incremental Control Algorithms

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

$$\Delta \boldsymbol{w} = \alpha\big(G_t - \hat{q}(S_t, A_t; \boldsymbol{w})\big)\nabla_w \hat{q}(S_t A_t; \boldsymbol{w})$$

  - For TD(0), the target is the TD target $G_{t:t+1} = R_{t+1} + \gamma\, \hat{v}(s_{t+1}; \boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha\big(R_{t+1} + \gamma\, \hat{v}(S_{t+1}; \boldsymbol{w}) - \hat{q}(S_t, A_t; \boldsymbol{w})\big)\nabla_w \hat{q}(S_t A_t; \boldsymbol{w})$$

Policy-Based Reinforcement Learning

- Previously we approximated the value or action-value function using parameters w

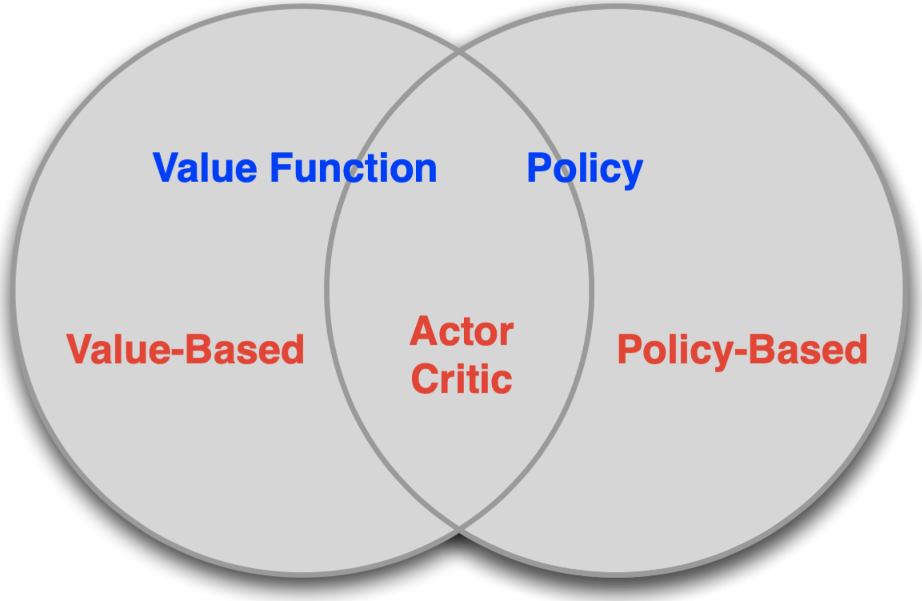
$$\mathrm{v}_w(s) \approx \mathrm{v}_\pi(s)$$
$$\mathrm{q}_w(s, a) \approx q_\pi(s, a)$$

- A policy was generated directly from the value function
  - e.g., using ε-greedy
- We now directly parametrize the policy

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s \,; \theta]$$

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



[David Silver, IRL, UCL 2015]

Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties (in contrast to value function approximation that can oscillate in some configurations)
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

Policy Optimization

- Policy based reinforcement learning is an optimization problem
- Find θ that maximizes $J(\theta)$
- Many possible optimization approaches
  - Gradient-free (e.g., Hill climbing, Genetic algorithms, etc.)
  - Gradient-based (e.g., Gradient descent)
- We focus on gradient descent, many extensions possible

Policy Gradient

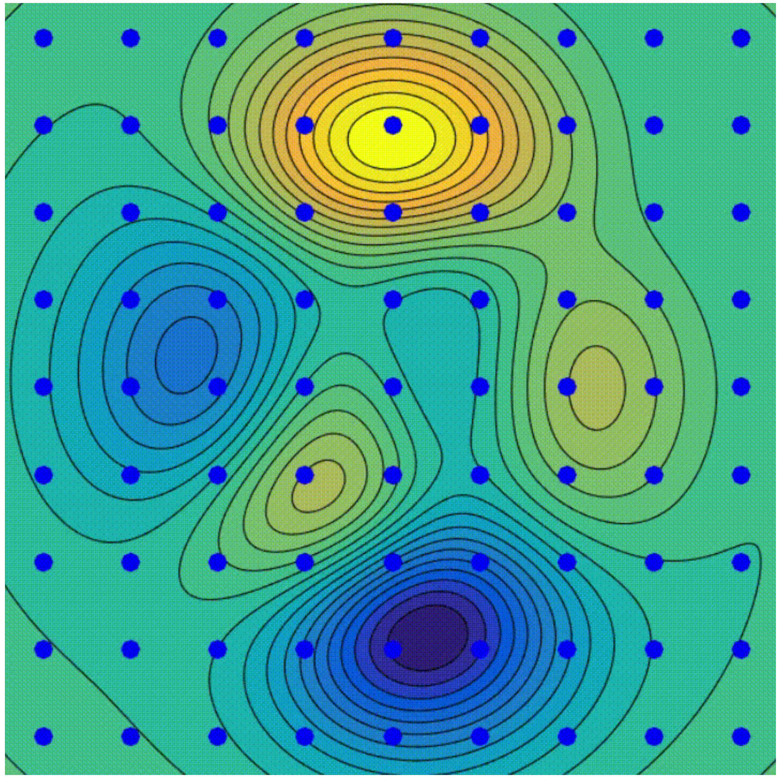
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\Delta\theta = a\nabla_{\theta}J(\boldsymbol{\theta})$$

- Where $\nabla_{\theta}J(\boldsymbol{\theta})$ is the policy gradient

$$\nabla_{\theta}J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter



Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_1(\theta) = \mathrm{V}^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[\mathrm{v}_1]$$

- In continuing environments, we can use the average value

$$J_{avV}(\theta) = \sum_{s} d^{\pi_\theta}(s)\, \mathrm{V}^{\pi_\theta}(s)$$

- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_{s} d^{\pi_\theta}(s) \sum_{a} \pi_\theta(s, a) \mathcal{R}_s^a$$

where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

One-Step MDPs

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_s^a$
- Use likelihood ratios to compute the policy gradient

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$
$$= \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)$$

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$

$$= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\, \mathcal{R}_s^a$$

$$\nabla_\theta J(\theta) = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\, \nabla_\theta \log \pi_\theta(s,a)\, \mathcal{R}_s^a$$
$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, r]$$

Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $q_\pi(s,a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem (Policy Gradient)

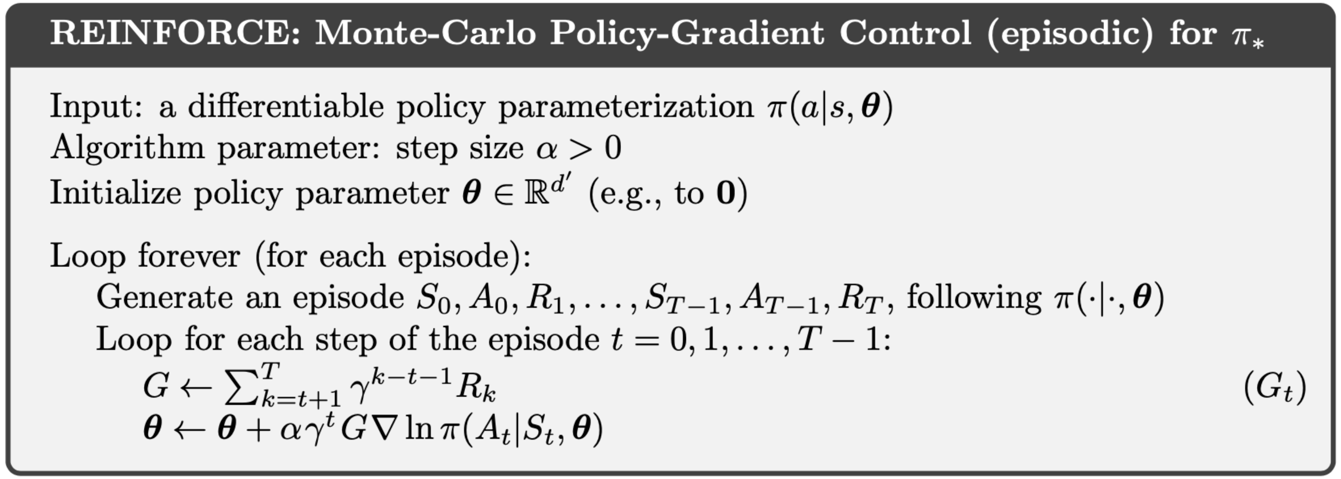
For any differentiable policy $\pi_\theta(s,a)$ and for any of the policy objective functions $J = J_1, J_{avR}$ or $\frac{1}{1-\gamma} J_{avV}$ the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) q_\pi(s,a)]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return G_t as an unbiased sample of $q_{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t)\, G_t$$



REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

```
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$                                        $(G_t)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
```

[An Introduction to Reinforcement Learning, Sutton and Barto]

Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function

$$q_w(s,a) \approx q_{\pi_\theta}(s,a)$$

- Actor-critic algorithms maintain two sets of parameters
  - Critic Updates action-value function parameters w
  - Actor Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) q_w(s,a)]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) q_w(s,a)$$

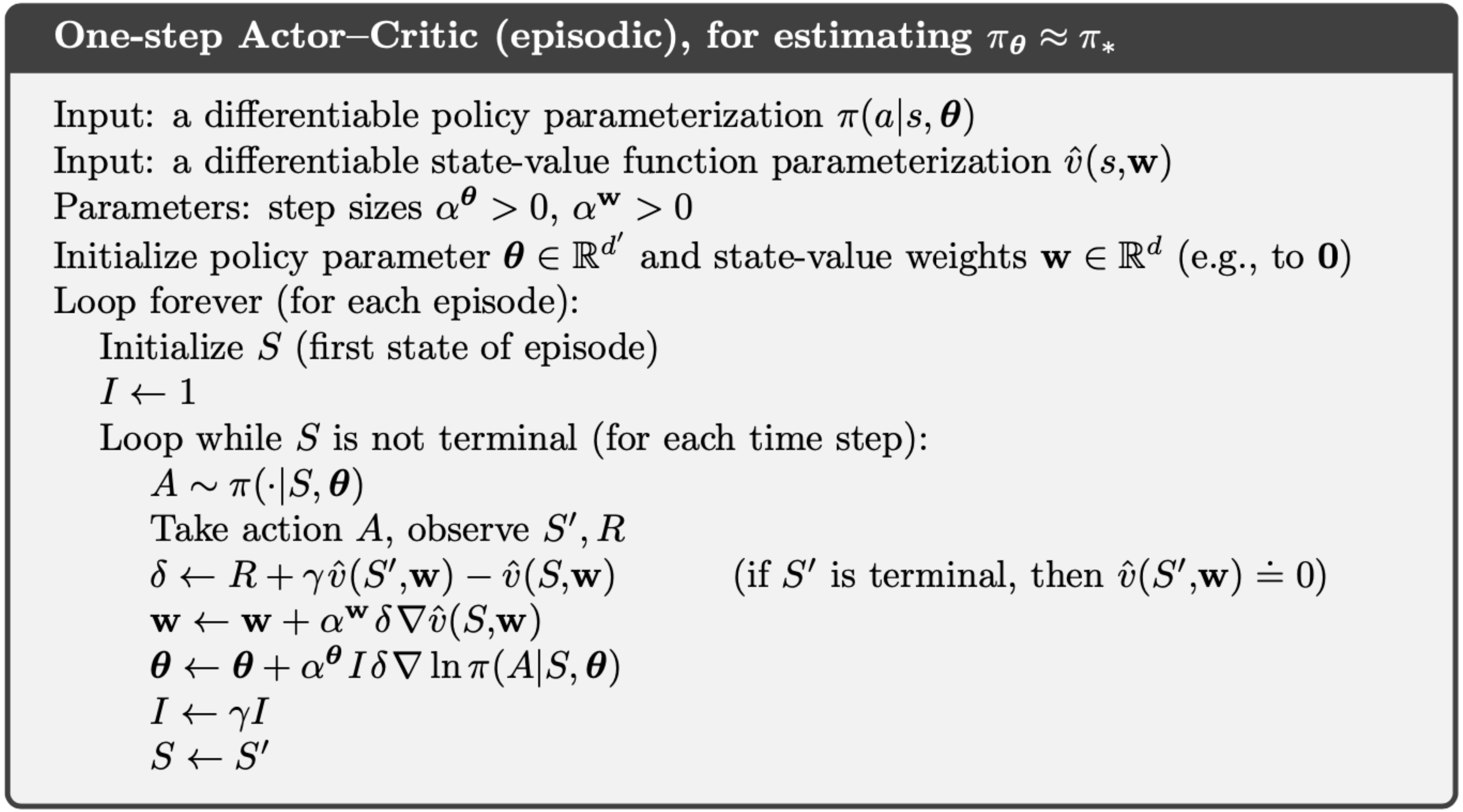
Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation (prediction)
- How good is policy π_θ for current parameters θ?
- Could also use e.g., least-squares policy evaluation

Action-Value Actor-Critic (1/2)

- Simple actor-critic algorithm based on action-value critic
- Using linear value function approximation $q_w(s,a) = \varphi(s,a)^T w$
  - Critic Updates w by linear TD(0)
  - Actor Updates θ by policy gradient

Action-Value Actor-Critic (2/2)



One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        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 $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln\pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

[An Introduction to Reinforcement Learning, Sutton and Barto]

Approximation with Deep Networks

- So far, the feature representation was typically “fixed”
- The parametrised functions $\hat{v}(s; w)$/ $\pi_\theta(s, a)$ were linear mappings of input features
- More complicated non-linear mappings are needed to generalize to more complex domains
- Popular choice is to use deep neural networks
  - Known to discover useful feature representation tailored to the specific task
  - We can leverage extensive research on architectures and optimisation from Supervised Learning

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”)

Stochastic Gradient Descent with Experience Replay

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

$$\mathcal{D} = \left\{\langle s_1, \mathrm{v}_1^{\pi}\rangle, \left\langle\langle s_2, \mathrm{v}_2^{\pi}\rangle\right\rangle, \dots, \left\langle\langle s_{\mathrm{T}}, \mathrm{v}_{\mathrm{T}}^{\pi}\rangle\right\rangle\right\}$$

- Repeat:
  1. Sample state, value from experience

$$\langle s, \mathrm{v}^{\pi} \rangle \sim \mathcal{D}$$

  2. Apply stochastic gradient descent update

$$\Delta \boldsymbol{w} = \alpha(\mathrm{v}_{\pi}(s) - \hat{\mathrm{v}}(s; \boldsymbol{w}))\nabla_{w}\hat{\mathrm{v}}(s; \boldsymbol{w})$$

- Converges to least squares solution

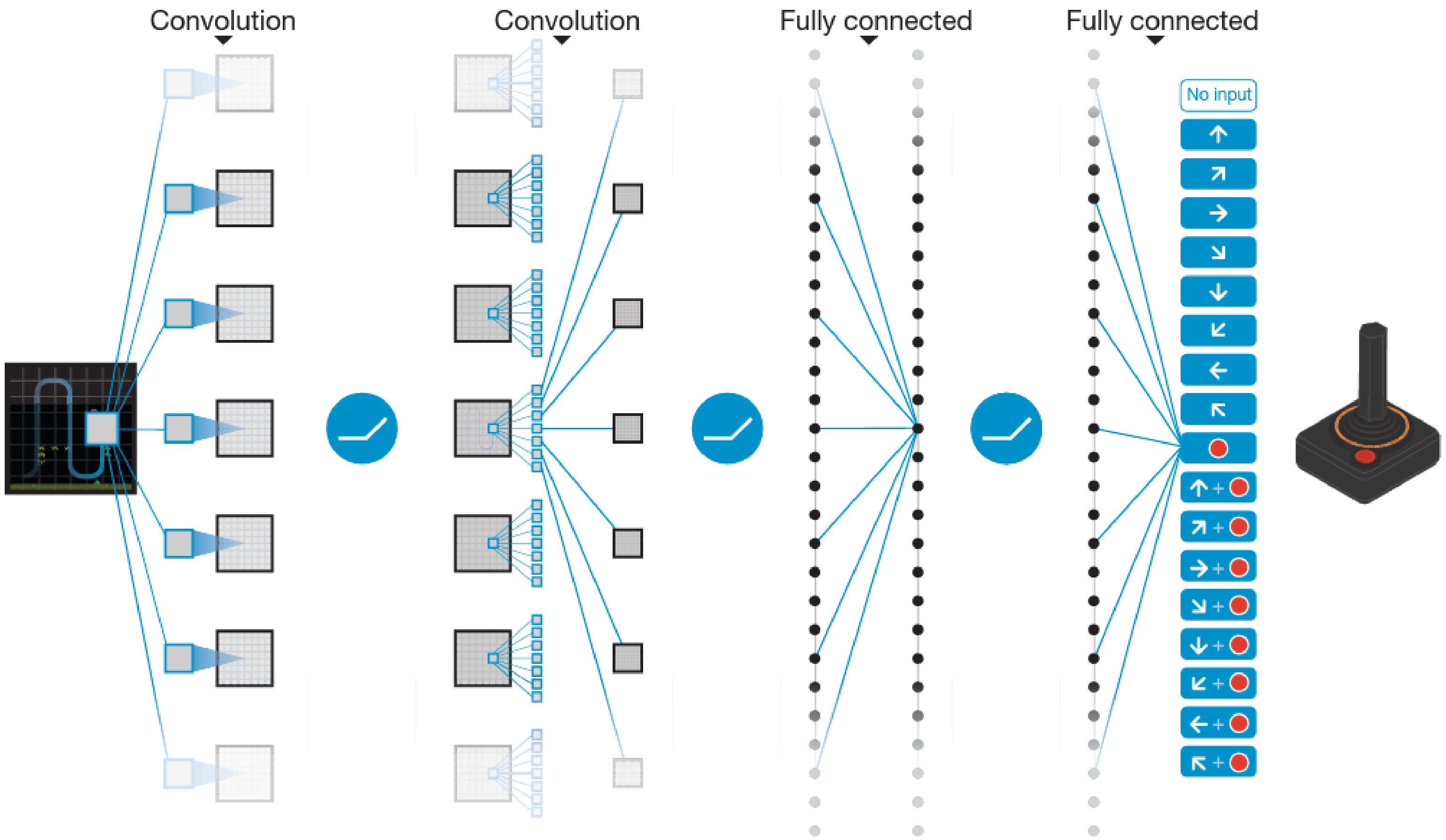
$$\boldsymbol{w}^{\pi} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, LS(\boldsymbol{w})$$

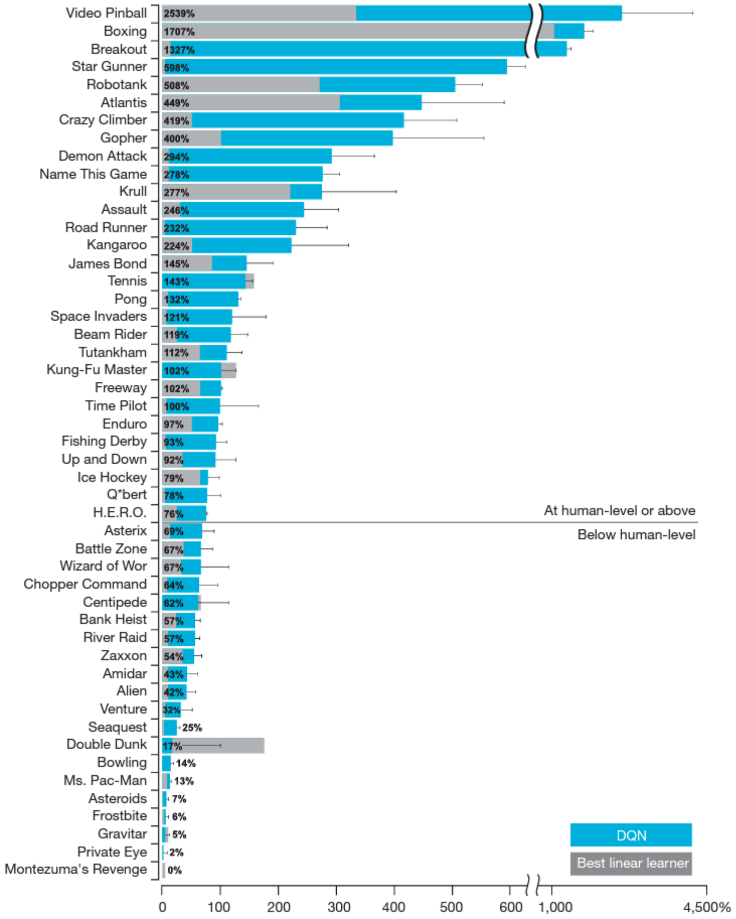
Experience Replay in Deep Q-Networks (DQN)

- Example of DQN that uses experience replay and fixed Q-targets
  - Take action a_t according to ε-greedy policy
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
  - Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
  - Compute Q-learning targets w.r.t. the parameters of a DQN w^-
  - Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

DQN in Atari





[doi:10.1038/nature14236]

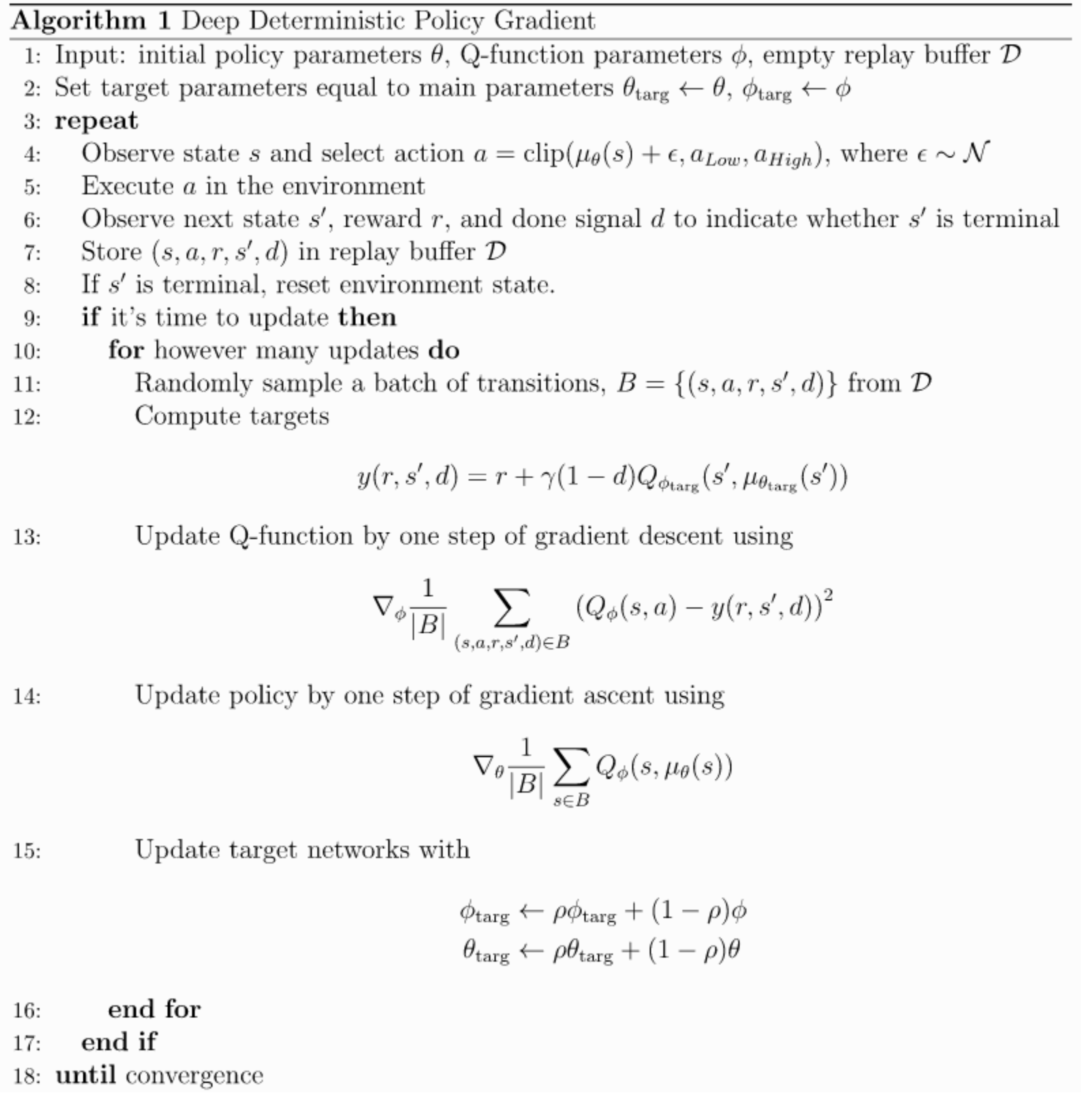
Continuous Action Spaces

- Vanilla DQN can’t be used for continuous action spaces (CAS)
  - Too many outputs to parametrize with a neural network
  - Discretization would be needed (suboptimal)
- Deep Deterministic Policy Gradient (DDPG) extends DQN to CAS
  - DDPG is an Actor Critic algorithm with critic π_θ and actor Q_w networks
  - Policy is deterministic, noise added for exploration $a_t = \pi_\theta(s_t) + \varepsilon$
  - Update the critic with the actor as max and then the actor to maximize the critic

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}[(r + \gamma Q(s', \pi(s'; \theta_i^-); w_i^-) - Q(s,a;w_i))^2]$$

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s \sim \mathcal{D}_i}[Q(s, \pi(s;\theta); w_i)]$$

DDPG



Algorithm 1 Deep Deterministic Policy Gradient

```
Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
    Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
    Execute $a$ in the environment
    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
    If $s'$ is terminal, reset environment state.
    if it's time to update then
        for however many updates do
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute targets
                $y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$
            Update Q-function by one step of gradient descent using
                $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$
            Update policy by one step of gradient ascent using
                $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$
            Update target networks with
                $\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi$
                $\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$
        end for
    end if
until convergence
```