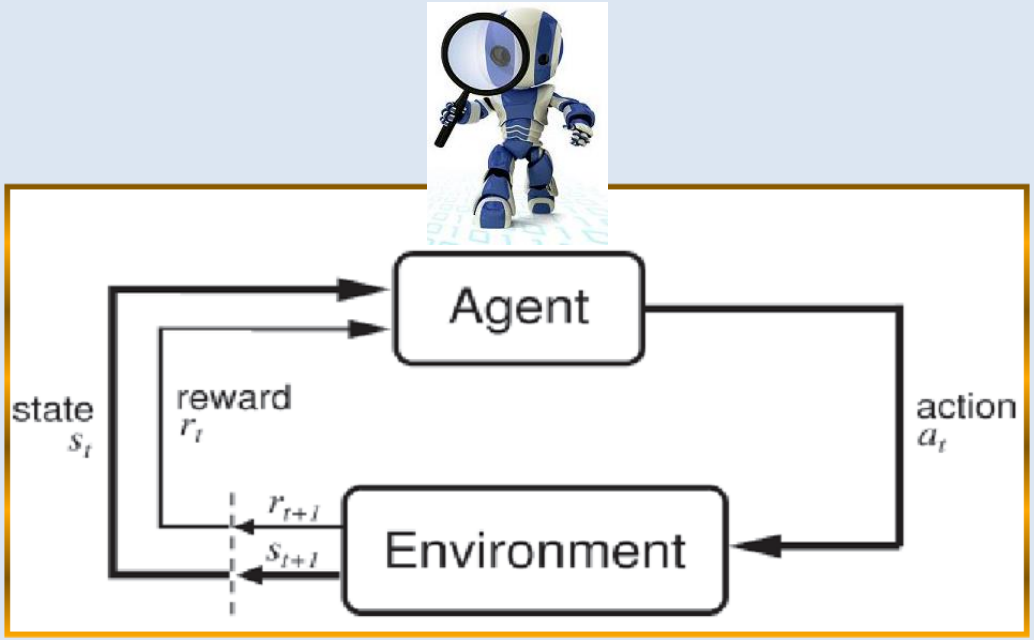
Machine Learning

Reinforcement Learning

Machine Learning

- **Supervised Learning**
  - **Teacher** tells learner what to remember
- **Reinforcement Learning**
  - **Environment** provides hints to learner
- **Unsupervised Learning**
  - **Learner** discovers on its own

➢Reinforcement Learning



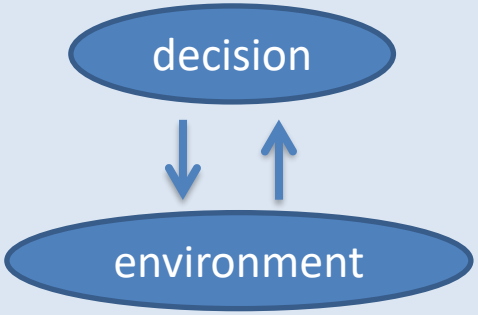
Special case of supervised learning where the desired decision (output) is unknown. Learning is performed through interaction with the environment using ***reward/penalty*** by maximizing the expected cumulative reward

Learning (Psychology)

Reinforcements for training animals

Negative reinforcements:
Pain and Hunger

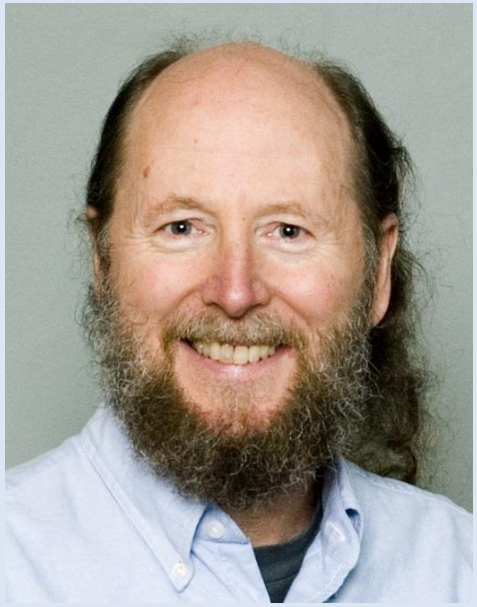
***Positive reinforcements*:**
Pleasure and food



Operant conditioning (***Ivan Pavlov, 1927***)
process by which humans and animals ***learn*** to behave in such a way to obtain ***rewards*** and avoid ***punishments***

Computational neuroscience
Hebbian learning (1961): synaptic weights between neurons are reinforced by simultaneously activation.

Reinforcement Learning



Richard. S. Sutton

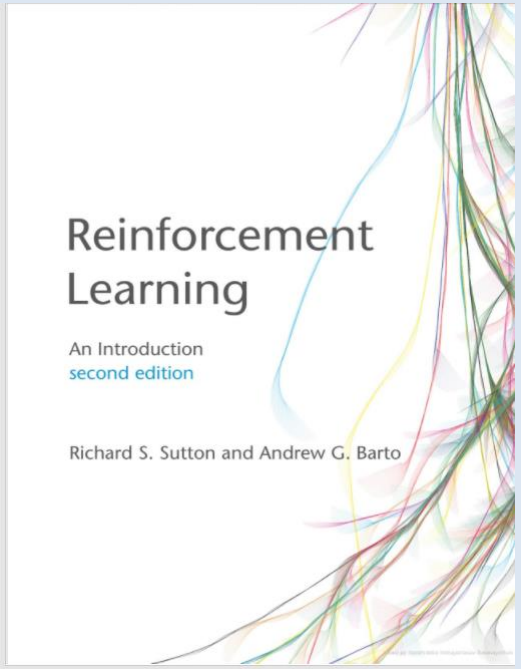
Professor and iCORE chair

Department of Computing Science

University of Alberta

Canada

(Psychology & Computer Science)



Richard S. Sutton and Andrew G. Barto

2nd edition 2018 (new edition 2020)

MIT Press, Cambridge, MA

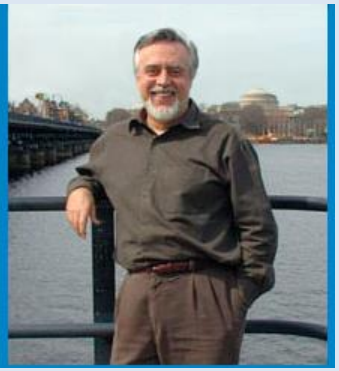
http://incompleteideas.net/book/the-book-2nd.html

43200 citations !!!!! (today)

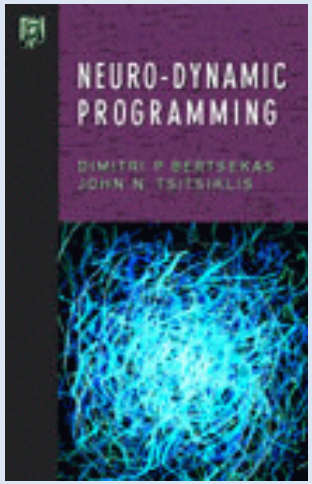
Bishop's book has 51500 citations !!!!!

Reinforcement Learning

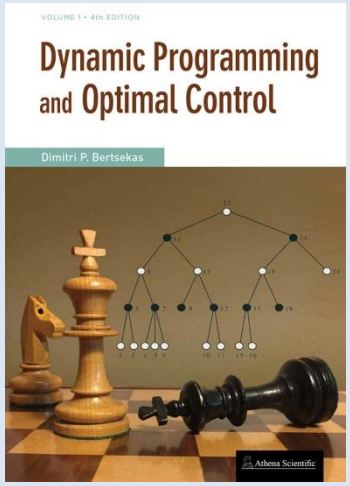
Dimitri P. Bertsekas
McAfee Professor of Engineering
Lab. for Information and Decision Systems
Room 32-660D
Massachusetts Institute of Technology
Cambridge, MA 02139
dimitrib@mit.edu



112000 citations in total



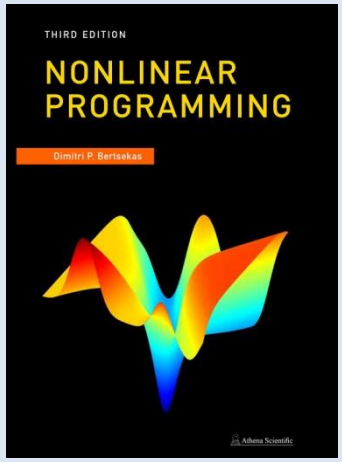
1996



2017

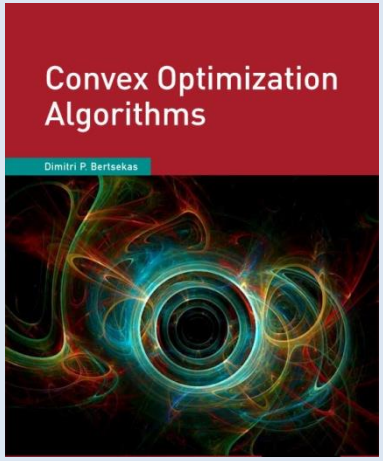
John N. Tsitsiklis
Massachusetts Institute of Technology
77 Massachusetts Avenue, 32-D784
Cambridge, MA 02139-4307, U.S.A.
+1-617-253-6175
jnt@mit.edu

57706 citations in total



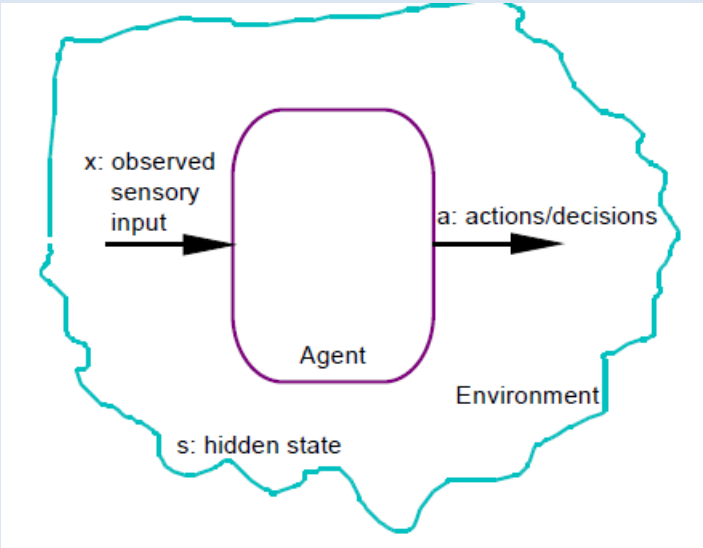
1999, 2015

(16700 citations)



2015

Intelligent Behavior



- **Agent** receives sensory input and take actions in environment
- Assume the agent receives **reward** (or penalties/losses)
- The goal is to **maximize the rewards** it receives (or minimize the losses)
- Choosing **actions that maximizing rewards (minimize losses)** is equivalent to **behave optimally**

Characteristics of Reinforcement Learning

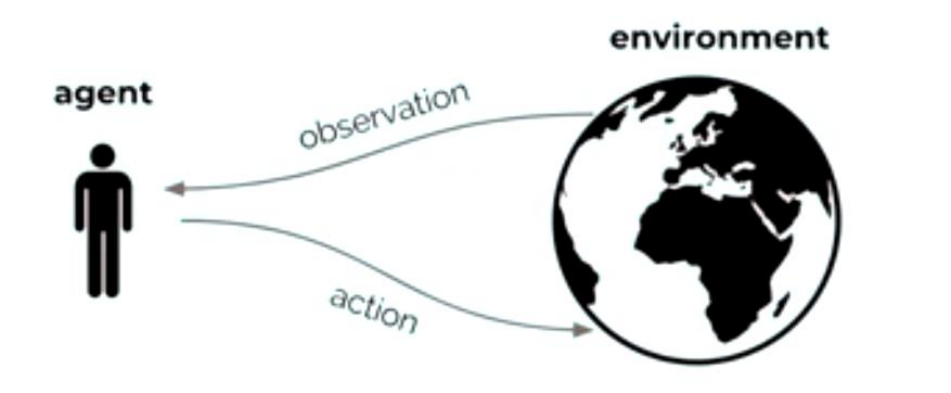
What makes reinforcement learning different from other machine learning paradigms?

- There is **no supervisor**, only a reward signal
- Learn by interacting with environment
  - active learning (not passive)
  - Interactions are sequential
  - Time really matters (sequential, non i.i.d data)
- Feedback is delayed, not instantaneous
- Goal - directed
- Agent's action affect the subsequent data it receives
- Can learn without examples of optimal behaviour

Agents with Intelligent Behavior

- **Game-playing**: Sequence of moves to win a game (e.g. Chess, Backgammon)
- **Robot** in a maze: Sequence of action to find a goal (e.g. position or object)
- **Autonomous vehicles** control (driving - navigation)
- **Routing problems:** Medical trials / Packets / Ads placement
- Manage an **investment portfolio**
- Learning to choose actions to **optimize factory output** (procedures)
- Control a **power station**
- **Recommendation systems** (lists)
- **……. (many more) …….**

Agent and Environment

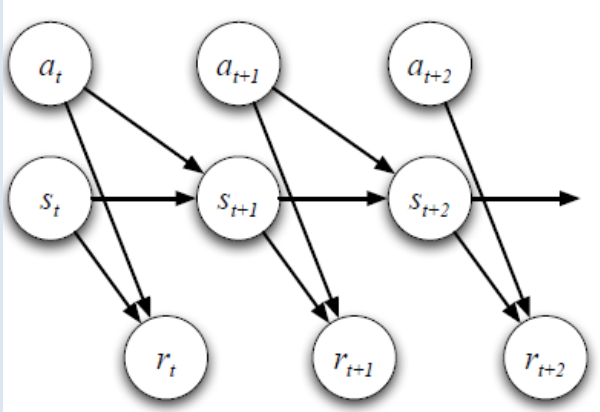


- At each step *t* the agent
  - Receives observation O_t and Reward r_t
  - Executes (*decides*) action a_t

- The environment
  - Receives the decision (action) a_t
  - Emits next observation O_{t+1} and reward R_{t+1}

Markov Decision Processes - MDPs

- Model of the agent-environment system covering the **Markov property**.



- An **MDP** is a tuple {***S, A, P, r, γ***}
  - ***S***: finite set of states
  - ***A***: finite set of actions
  - ***P***: state transition probability function $P_{ss'}^{a} = P\left(s_{t+1} = s' \mid s_t = s, a_t = a\right)$
  - ***r***: reward function $r(s,a) = E\left\{r_{t+1} \mid s_t = s, a_t = a\right\}$
  - ***γ***: discount factor $\gamma \in [0,1]$

States (s)

- Captures whatever **information is available to the agent** at step t about its **environment**.
- These are **structures built up over time** from **sequences** of sensations, memories, etc.
- **Markovian property:** We could throw away the history once state is known

$$\mathrm{P}\{r_{t+1} = r, s_{t+1} = s' | s_0, a_0, r_1, \ldots, s_{t-1}, a_{t-1}, r_t, s_t, a_t\} =$$
$$\mathrm{P}\{r_{t+1} = r, s_{t+1} = s' | s_t, a_t\}$$

Reward

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t – defines the goal
- The agent's goal is to maximize cumulative reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

which is called return

- RL is based on the reward hypothesis:

Any goal can be formalized as the outcome of maximizing a cumulative reward

Policy (π)

- It defines the agent's behaviour, a **decision mechanism**
- A **map** from states to actions, S $\rightarrow$ A

- Deterministic policy

$$\pi(s) = a$$

- Stochastic decision: probability distribution

$$\pi(a|s) = P(a_t = a|s_t = s)$$

Value Functions

- **State** value function

$$V^{\pi}(s) = E(G_t|S_t = s, \pi) =$$
$$E\left(R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, R_{t+3} + \ \ldots |S_t = s, \pi\right) =$$
$$E\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\, |S_t = s, \pi\right)$$

- Determines how good is it for agent to be in a given state,
- Gives the long-term value of state s: it is a prediction of future reward
- The value depends on a policy (select between actions)

- **State-action** value function

$$Q^{\pi}(s,a) = E_{\pi}(G_t | s_t = s, a_t = a)$$

$$= E_{\pi}(R_{t+1} + \gamma\, R_{t+2} + \gamma^2 R_{t+3} + \ \ldots | s_t = s, a_t = a)$$

$$= \ E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, | s_t = s, a_t = a\right)$$

- Determines how good is it to perform an action from a given state and then follow policy
- It is the expected return starting from state s, taking action a, and then following policy π

Bellman Equations

Richard Bellman, 1957

(*600 papers, 35 books, 7 monographs)*

- Bellman equations expresses the relationship between the values of a state s and the values of its successor states

- The value of the next state must equal the discounted value of the expected next state, plus the reward expected along the way (recursive form)

Bellman Expectation Equation (V)

$$V^{\pi}(s) = E_{\pi}(R_t \mid s_t = s) = E_{\pi}\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s \right)$$

$$= E_{\pi}(r_{t+1} + \gamma R_{t+1} \mid s_t = s)$$

$$= E_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s) = r_{t+1} + \gamma V^{\pi}(s_{t+1})$$

The value function is decomposed into **2 parts**

- Immediate **reward** r_{t+1}
- Discounted value of successor state $\gamma V(s_{t+1})$

Bellman Expectation Equation (Q)

- The **state-action value function** can be similarly decomposed into **2 parts**

$$
\begin{aligned}
Q^{\pi}(s,a) &= E_{\pi}\left(R_t \mid s_t = s, a_t = a\right) \\
&= E_{\pi}\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s, a_t = a \right) \\
&= E_{\pi}\left(r_{t+1} + \gamma Q^{\pi}\left(s_{t+1}, a_{t+1}\right) \mid s_t = s, a_t = a\right) \\
&= r_{t+1} + \gamma Q^{\pi}\left(s_{t+1}, a_{t+1}\right)
\end{aligned}
$$

Optimal Policies and Values

- **Optimal state value function**:

$$V^*(s) = \max_{\pi} V^{\pi}(s) = \max_{a} \left\{ r(s,a) + \gamma \sum_{s'} P_{ss'}^{a} V^*(s') \right\}$$

- If policy π is such that in each state s it **selects an action that maximize value**, then π is an **optimal policy**

- An **optimal policy** can be found **by using greedily the V*(s)**

- **Optimal state-action value function:**

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall s$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P_{ss'}^{a} \max_{a'} Q^*(s',a')$$

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

- An **optimal policy** can be found by maximizing over the Q*(s,a), i.e.

$$\pi^*(a \mid s) = 1 \text{ if } a = \arg\max_{a \in A} \{Q^*(s,a)\}$$

- There is always a deterministic optimal policy for any MDP
- If we know *Q*(s,a)* we have the optimal policy

Exploration and Exploitation

- **Exploration** finds more information about the environment to (possibly) makes a better decision
- **Exploitation** makes the best decision given current information (exploits known information to maximize reward)
- Both of them are important. Fundamental problem not occurring in supervised learning
- A trade-off is needed between exploration and exploitation

Strategies

- **ε-greedy:** With probability *ε* choose one action at random (uniformly), and choose the best action with probability *1-ε* (*ε* is gradually reduced)

- **Probabilistic:** Use probabilistic action selection *(soft-max)*

$$P(a \mid s) = \frac{e^{Q(s,a)/T}}{\sum_{b=1}^{A} e^{Q(s,b)/T}}$$

Reinforcement Learning Methods

- Model-free prediction
- Estimate the value function of an unknown MDP

- Various methodologies
  1. **Monte-Carlo (MC) - Sample-based** reinforcement methods
  2. **Temporal-Difference (TD)** methods
  3. **Value function Approximation**

1. Monte Carlo (MC) methods

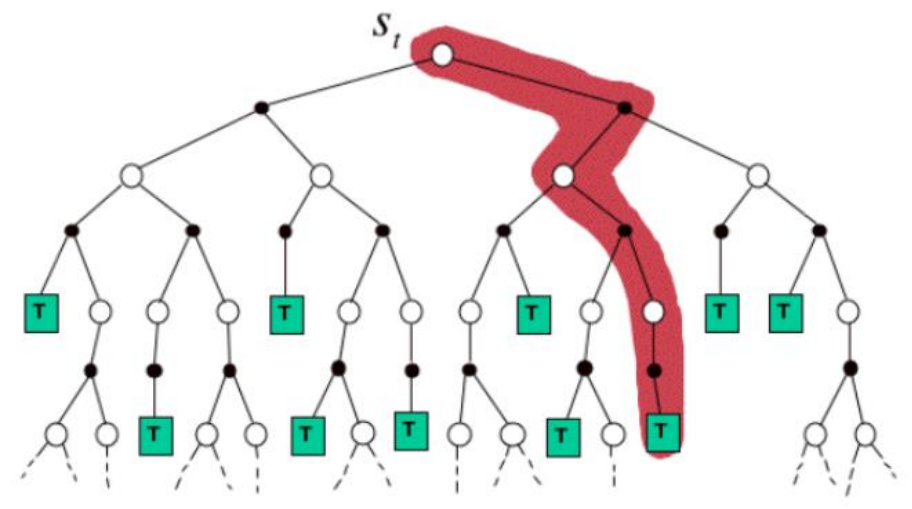
- MC learns *directly* from episodes of experience:
  - sample sequences of *states, actions, rewards* (*s,a,r*)
- MC is *model-free*: no knowledge of model
  - (MDP transitions/rewards)
- Learns from *complete* episodes (no bootstrapping)
  - all episodes must terminate
- Simplest idea: value = mean return,
  - i.e. solve problems by **averaging** sample returns

- **Return** is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

- **Value function** is the expected return

$$V^{\pi}(s) = E(G_t |\ S_t = s, \pi)$$



- Increment total return

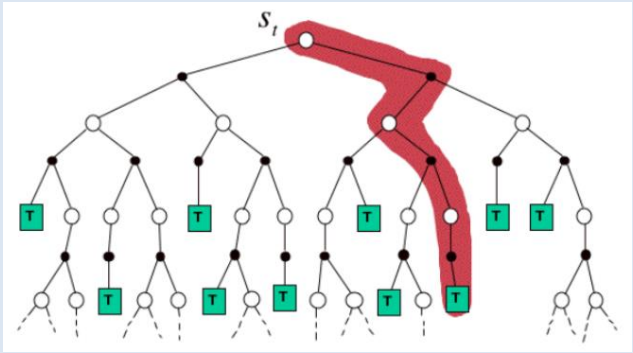
$$S(s_t) \leftarrow S(s_t) + G_t$$

$$V(s_t) \leftarrow \frac{S(s_t)}{n(s_t)}$$

n(s) frequency of state s, and $\boldsymbol{G_t}$ the actual return following s_t

- Update V(s) incrementally after episode

- For each state s_t with return G_t



$$N(s_t) = N(s_t) + 1$$

$$V(s_t) \leftarrow V(s_t) + \frac{1}{n(s_t)}[G_t - V(s_t)]$$

$$V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]$$

2. Temporal Difference (TD) Learning

- TD earns directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions/rewards is required
- TD **learns** from incomplete episodes (*bootstrapping*)
- Based on Bellman equations:

$$V^{\pi}(s) = E_{\pi}(r_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s)$$

$$V^{\pi}(S_t) = r_{t+1} + \gamma V^{\pi}(S_{t+1})$$

Temporal Difference (TD) Learning

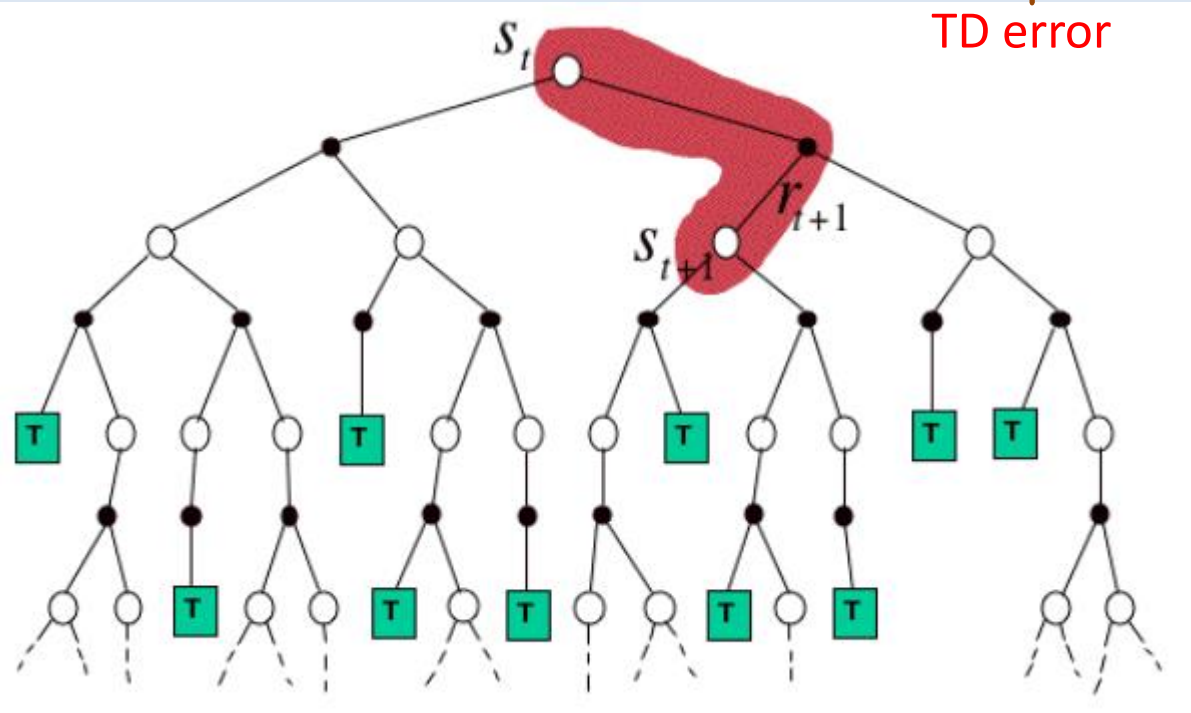
- **TD generic update rule:**

$$V^{\pi}(s_t)$$

TD target

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

TD error



TD(0) Algorithm for Learning V^{π}

- Initialise $V(s)$ arbitrarily; π is the policy to be evaluated; choose learning rate α and discount factor γ
- Repeat for each episode

  Pick a start state s

  Repeat for each step in episode

  Get action a given by policy π for state s

  Take action a, observe reward r and next state s'

  $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$ ←

  $s \leftarrow s'$ **Learning rate**

  until s is terminal

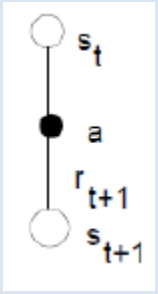
MC and TD

- **MC update:**

$$V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]$$

Actual return from s_t to end of episode

- **TD update:**



TD error

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

TD target

Estimate of the return according to the current policy

Q-Learning

(*Watkins, Ph.D. Thesis, Cambridge Univ. 1989*)

- Off-policy greedy method: evaluate or improve one policy while acting using another

- Learn **state-action** value functions *Q(s,a)*

$Q^{\pi}(s_{t+1}, a_{t+1})$

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

TD error

(or)

$$Q(s_t, a_t) \longleftarrow [1 - \alpha] Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a)]$$

Algorithm: Q-Learning

- Initialise $Q(s, a)$
- Repeat `many times`
  - Pick s `start state`
  - Repeat `each step to goal`
    - Choose a based on $Q(s, a)$ ϵ-`greedy`
    - Do a, observe r, s'
    - $Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    - $s = s'$
  - Until s terminal

SARSA (policy improvement)

- On-policy TD method: evaluate or improve the current policy used for control
- SARSA takes **exploration into account** in updates

~~$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right]$~~

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right]$$

Use the action actually chosen in updates (e-greedy)

3. Value Function Approximation

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

- Solution for large MDPs
  - Estimate value function with function approximation
  - Update function parameters using TD learning

Value Function Approximation

- Several function approximators can be used, e.g.
  - Neural Networks – Deep NNs
  - Linear model – Linear combinations of features
  - Kernel machines
  - Statistical regression models
  - Decision trees
  - …..

Value function approximation with Temporal Difference Learning

- Assume a parametric model (***w***) for the **Q-value** function

$$Q^{\pi}(s, \boldsymbol{a}) \approx \max_{a} Q(s, a; w)$$

- Estimate model parameters, *w*, according to *Q-learning* scheme (***gradient descent***):

$$E(w) = \frac{1}{2N}\sum_{t=1}^{N}(Q(s_t, a_t, w) - Q^{\pi}(s_t, a_t))^2 =$$

$$= \frac{1}{2N}\sum_{t=1}^{N}\left(Q(s_t, a_t, w) - \left(r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a, w)\right)\right)^2$$

Learning is executed on-line, or with mini-batches

- On-line learning at every iteration:

$$\delta_t = Q(s_t, a_t; w^{old}) - \left(r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a; w^{old}\right)\right)$$

$$w^{new} \longleftarrow w^{old} - \alpha\, \delta_t\, \nabla_w Q(s_t, a_t; w)$$

Experience replay memory

- Store the agent's experiences at each time step $e_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset $D = e_1, ..., e_n$ pooled over many episodes into a replay memory

- In practice, only store the last N experience tuples in the replay memory and sample uniformly from D when performing update

Double Q-learning strategy

- Collect samples and store them to memory D (by substitution)
- Learning is conducted using experience replay memory mini batches
- Target is calculated from a clone (target) Q-function
- After performing some learning epochs, copy updated Q-network (function) to target Q-network (function)

$$D = \{s_t, a_t, r_{t+1}, s_{t+1}\}$$

$$\delta_t = Q(s_t, a_t; w) - (r_{t+1} + \gamma \max_a Q(s_{t+1}, a; w^{target}))$$

$$w^{new} \longleftarrow w - \alpha\, \delta_t\, \nabla_w Q(s_t, a_t; w)$$

use mini baches S $\subseteq$ D

$$w^{target} = w^{new}$$

Linear Value Function Approximation

- Represent value function as a linear combination of features
- Describe state *s* as a **feature vector**

$$\phi(s) = \left(\phi_1(s), \phi_2(s), \dots, \phi_n(s)\right)$$

- Or state-action feature vector

$$\phi(s,a) = \left(\phi_1(s,a), \phi_2(s,a), \dots, \phi_n(s,a)\right)$$

- Then the value function can be any **regression model**

  e.g. linear regression model

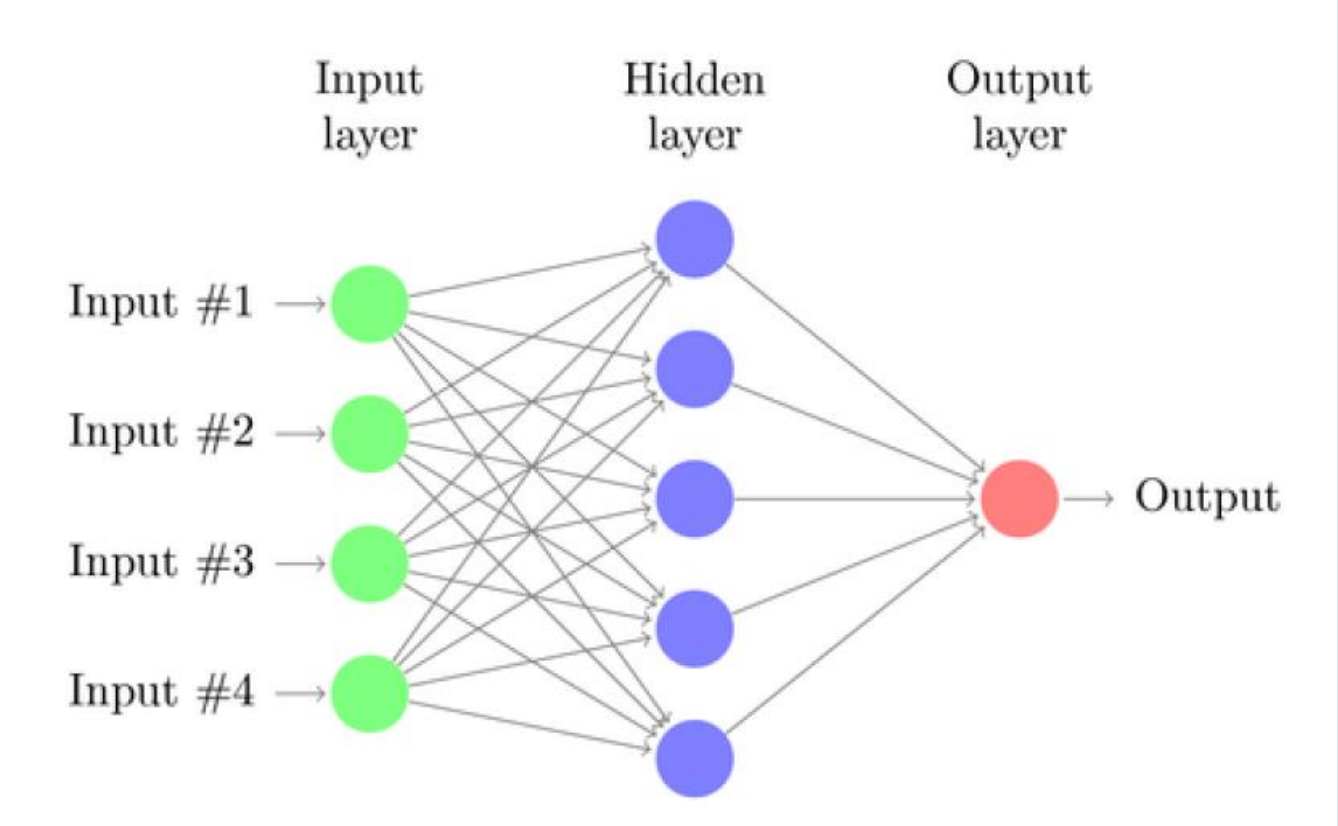
$$V(s) = w^T \phi(s) = w_1 \phi_1(s) + w_2 \phi_2(s) + \cdots + w_n \phi_n(s)$$

$$Q(s_t, a_t, w) = w^T \phi(s_t, a_t) = \sum_{i=1}^{n} w_i \phi_i(s_t, a_t)$$

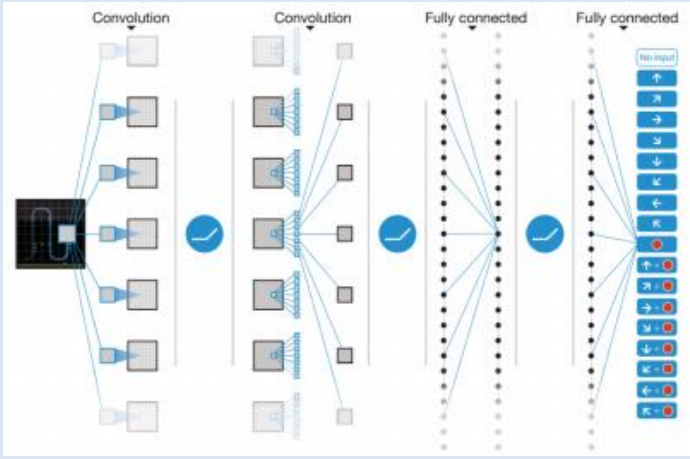
- w_i are linear weights

Neural Networks for value function approximation

- Value function has a neural network (non-linear) design
- Model parameters, ***w***, are the weights of network



Deep Q learning (DQN)

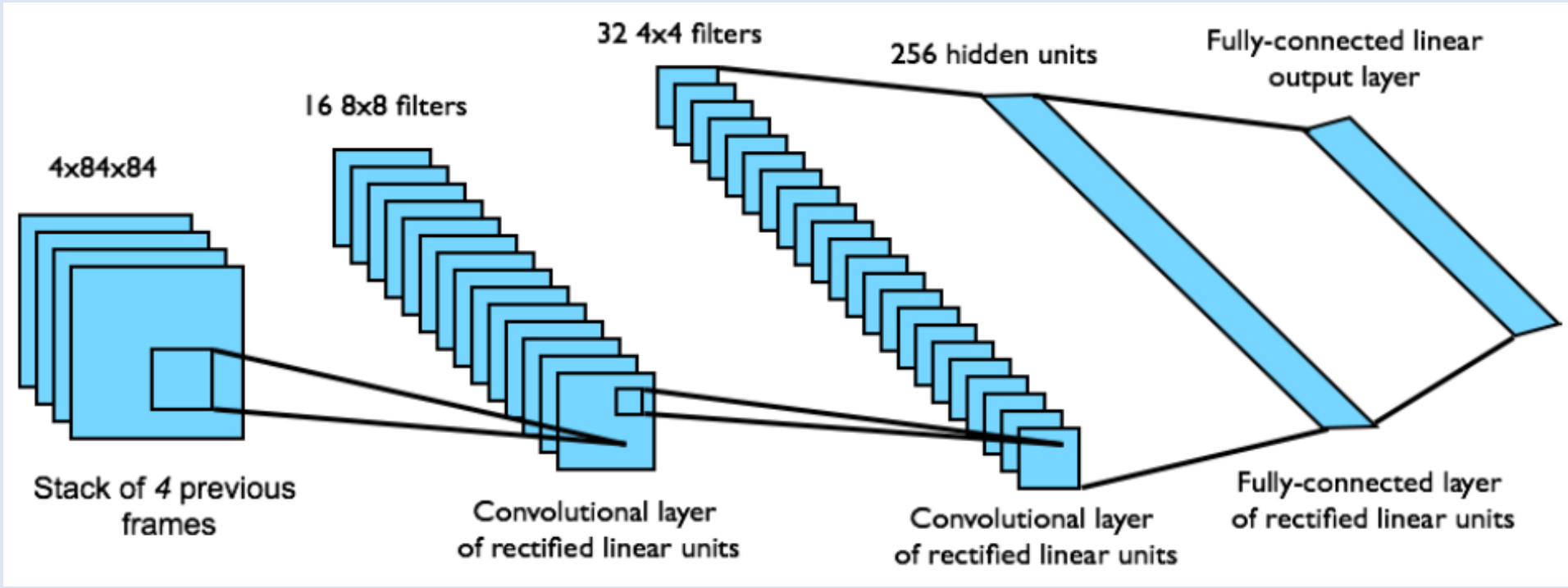


- Use deep net to estimate Q-values
- Input: the state of agent
- Output: Q-values for possible actions
- Learning step: gradient descent with the loss
- Policy: choose action to maximize the Q-value

Deep Reinforcement Learning
(Deep Mind Tech., Google - 2015)

Towards General Artificial Intelligence

- **Playing Atari with Deep Reinforcement Learning**. *ArXiv (2013)*
  - 7 Atari games
  - The first step towards "General Artificial Intelligence"
- DeepMind got acquired by @Google (2014)
- **Human-level control through deep reinforcement learning**. *Nature (2015)*
  - 49 Atari games
  - Google patented "Deep Reinforcement Learning"



- Network architecture and hyperparameters fixed across all games
- Input state is stack of raw pixels from last 4 frames
- Output is Q(s,a) got 18 joystick/button positions
- Reward is change in score for that step

Policy Gradient

- Assumption: Policy is parametric model $a = \pi_\theta(s)$
- Goal: Directly maximize the total expected reward over the entire trajectory, τ

$$J(\theta) = E_{\pi_\theta}[R(\tau)]$$

- Learning: Gradient descent on the policy's parameters θ

$$\hat{\theta} : \max_\theta J(\theta) \qquad \theta \leftarrow \theta + \nabla_\theta E_{\pi_\theta}[R(\tau)]$$

Policy Gradient

- Long-term reward:
  - sum of rewards for the trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T)$

$$R(\tau) = \sum_{t=1}^{T} r(s_t)$$

- **Value of policy** $J(\theta) = E_{\pi_\theta}[R(\tau)] = \sum_{\tau} P(\tau|\theta) R(\tau)$
  - $P(\tau/\theta)$: probability of trajectory following the policy π_θ

- **Goal**: find policy parameters θ that maximize $J(\theta)$

$$\hat{\theta} : \max_{\theta} J(\theta) = \max_{\theta} \sum_{\tau} P(\tau|\theta) R(\tau)$$

Computing the gradient

- (of course) we cannot compute all trajectories … but we can sample m trajectories

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_i R(\tau_i) \, \nabla_\theta \log P(\tau_i|\theta)$$

- *gradient*

$$\begin{aligned}
\nabla_\theta \log P(\tau|\theta) &= \nabla_\theta \log \left[\mu(s_0) \prod_{i=0}^{T-1} \pi_\theta(a_i|s_i) P(s_{i+1}|s_i, a_i) \right] \\
&= \nabla_\theta \left[\log \mu(s_0) + \sum_{i=0}^{T-1} \log \pi_\theta(a_i|s_i) + \log P(s_{i+1}|s_i, a_i) \right] \\
&= \sum_{i=0}^{T-1} \underbrace{\nabla_\theta \log \pi_\theta(a_i|s_i)}_{\text{No dynamics model required!}}
\end{aligned}$$

Computing the gradient

- Solution

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1,\ldots,m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{it}|s_{it})$$

- If action is discrete use (e.g.) Deep NN with softmax (last layer with so many neurons as actions)
- In continuous spaces of actions, action is directly generated
- Problem: gradient is noisy and has large variance
- Need to reduce variance

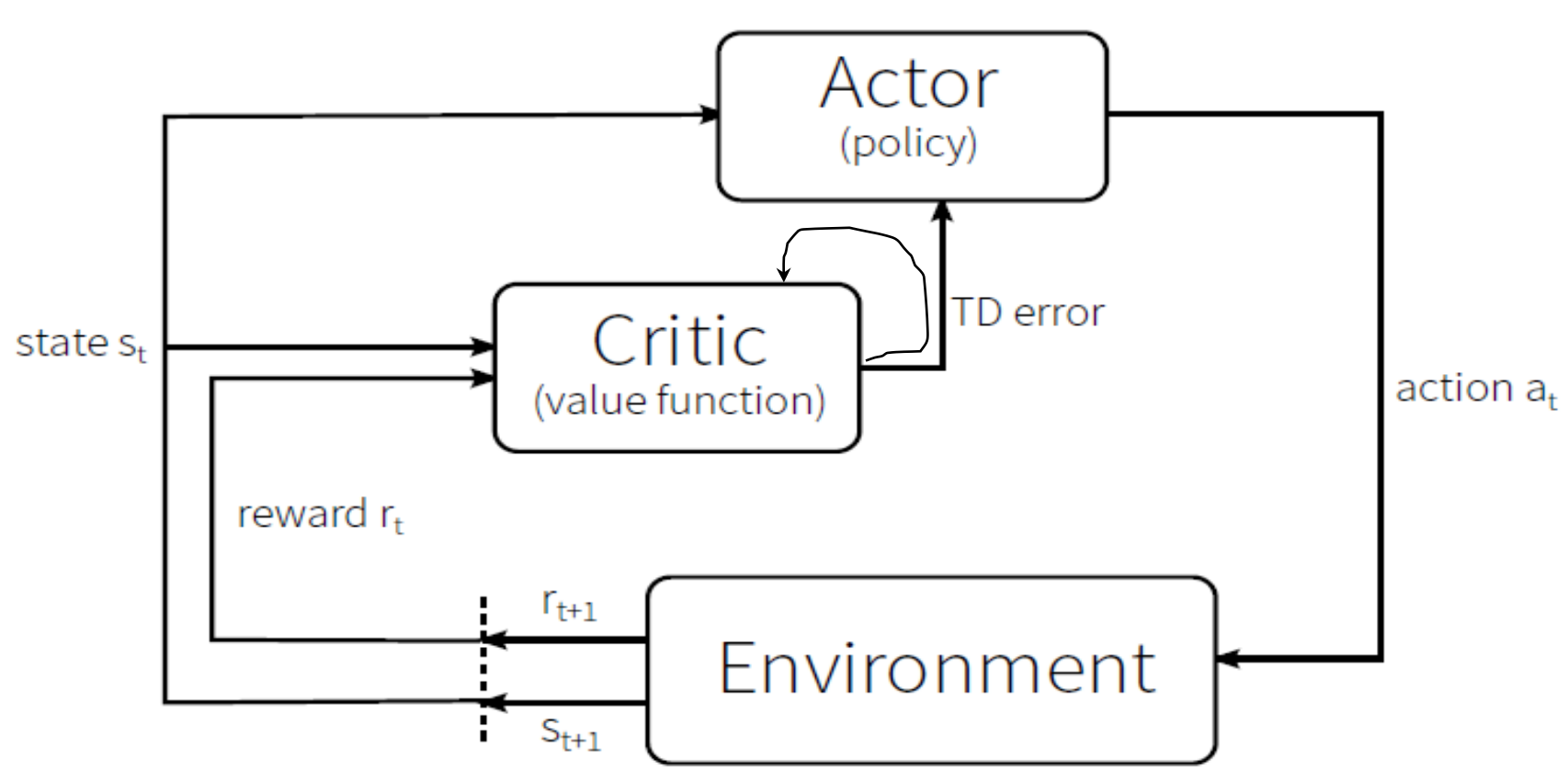
Monte-Carlo Policy Gradient

REINFORCE algorithm *(Williams, 1992)*

Given architecture with parameters θ to implement π_θ
Initialize θ randomly
repeat
  Generate episode $\{s_1, a_1, r_2, \ldots s_{T-1}, a_{T-1}, r_T, s_T\} \sim \pi_\theta$
  for all time steps $t = 1$ to $T - 1$ **do**
    Get $R_t \leftarrow$ long-term return from step t to T
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) R_t$
  end for
until convergence

Actor – Critic methods

- Learn Value function and Policy
- **Critic:** evaluates the current policy and the result is used in the policy training
- **Actor:** implements the policy and is trained using Policy Gradient in direction suggested by critic
- Have **separate memory structure** to represent the policy independent of the value function


Actor
(policy)
Critic
(value function)
TD error
state s_t
action a_t
reward r_t
r_{t+1}
Environment
s_{t+1}

Reduce variance with baseline

- R_t has a lot of variance
- We can reduce variance subtracting a baseline to the estimator

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b(s_t))$$

doesn't depend on actions taken

- A good baseline is value function

$$b(s_t) = V^{\pi_\theta}(s_t)$$

- Use another parametric model w

$$V^{\pi_\theta}(s_t) \approx V_w(s_t)$$

Monte-Carlo Actor Critic (with baseline function)

Given architecture with parameters θ to implement π_θ and parameters w to approximate V

Initialize θ randomly

repeat

  Generate episode $\{s_1, a_1, r_2, \ldots s_{T-1}, a_{T-1}, r_T, s_T\} \sim \pi_\theta$

  for all time steps $t = 1$ to $T - 1$ **do**

    Get $R_t \leftarrow$ long-term return from step t to T

    $\delta \leftarrow R_t - V_w(s_t)$

    $w \leftarrow w + \beta\delta\nabla_w V_w(s_t)$

    $\theta \leftarrow \theta + \alpha\delta\nabla_\theta \log \pi_\theta(a_t|s_t)$

  end for

until convergence

Estimating the TD error

- **Critic**: estimate the state-action value

$$Q_w(s, a; \boldsymbol{w}) \approx Q^{\pi_\theta}(s, a)$$

- **Actor**: policy evaluation (action selection)

$$\pi_\theta(s, a) \approx \pi(s, \boldsymbol{a}; \boldsymbol{\theta})$$

- TD error

$$\delta^{\pi_\theta} = \left(r + \gamma Q_w(s', a')\right) - Q_w(s.a)$$

One step Actor Critic

Given architecture with parameters θ to implement π_θ and parameters w to approximate Q
Initialize θ randomly
repeat
  Set s to initial state
  Get a from π_θ
  repeat
    Take action a and observe reward r and new state s'
    Get a' from π_θ
    $\delta \leftarrow r + Q_w(s', a') - Q_w(s, a)$ // TD-error (Bellman equation)
    $w \leftarrow w + \beta\delta\nabla_w Q_w(s, a)$ // critic update
    $\theta \leftarrow \theta + \alpha\nabla_\theta \log \pi_\theta(a|s) Q_w(s, a)$ // Actor update
    $s \leftarrow s'$
  until s is terminal
until convergence

Actor-Critic schemes: TD and Advantage

- TD error $\delta^{\pi_\theta} = \left(r_{t+1} + \gamma\, V^{\pi_\theta}(s_{t+1})\right) - V^{\pi_\theta}(s_t)$

- Advantage function (Critic estimate the advantage function)

$$E_{\pi_\theta}(\delta^{\pi_\theta}|s,a) = E_{\pi_\theta}\left[r_{t+1} + \gamma\, V^{\pi_\theta}(s_{t+1})\,|s,a\right] - V^{\pi_\theta}(s) =$$
$$= Q_w(s,a) - V_w(s) = A_w(s,a)$$

  - Use **two** function approximators with **two** set of parameters, *w*, *v*

$$\begin{aligned} V^{\pi_\theta}(s) &\approx V_v(s) \\ Q^{\pi_\theta}(s,a) &\approx Q_w(s,a) \\ A(s,a) &= Q_w(s,a) - V_v(s) \end{aligned}$$

dueling Networks

  - update both functions