



LUND UNIVERSITY

# **FMAN-45: Machine Learning**

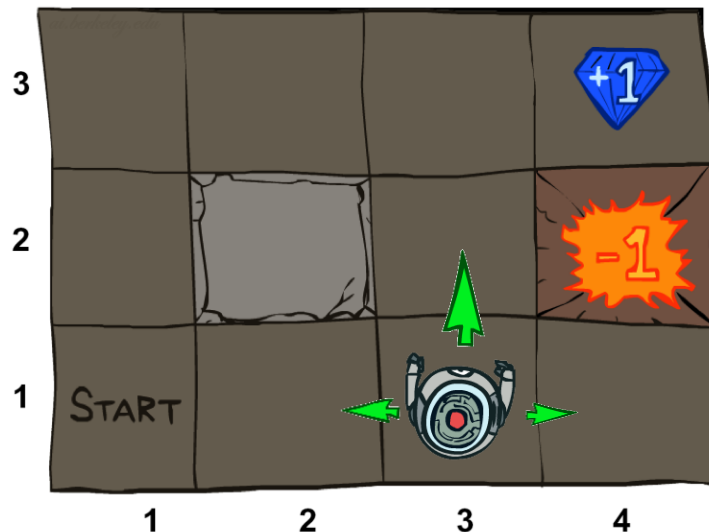
Lecture 12:

Reinforcement Learning II

*Aleksis Pirinen, Cristian Sminchisescu*

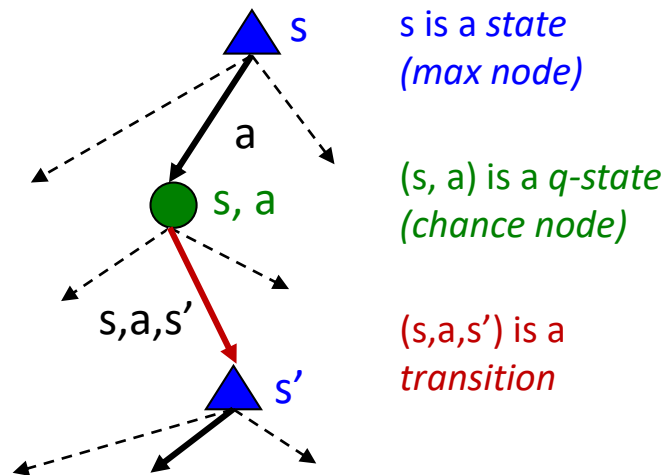
# Last Time: Markov Decision Processes (MDPs)

- An MDP is defined by:
  - A set of states  $s \in S$
  - A set of actions  $a \in A$
  - A transition function  $T(s, a, s')$ 
    - Probability that  $a$  from  $s$  leads to  $s'$ , i.e.,  $P(s' | s, a)$
    - Also called the *model* or the *dynamics*
    - Similar to successor function, except may be stochastic
  - A reward function  $R(s, a, s')$ 
    - Sometimes just  $R(s)$  or  $R(s')$
    - May be non-deterministic  $\rightarrow$  extend dynamics model  $P(s' | s, a)$  into  $P(r, s' | s, a)$  [although not very common]
  - A start state
  - Maybe a terminal state
- MDPs are non-deterministic search problems



# Last Time: Optimal Quantities

- The optimal value of a state  $s$ :  
 $V^*(s)$  = expected utility starting in  $s$  and acting optimally
- The optimal value of a Q-state  $(s,a)$ :  
 $Q^*(s,a)$  = expected utility starting out having taken action  $a$  from state  $s$  and (thereafter) acting optimally
- The optimal policy:  
 $\pi^*(s)$  = optimal action from state  $s$



# Last Time: The Bellman Optimality Equations

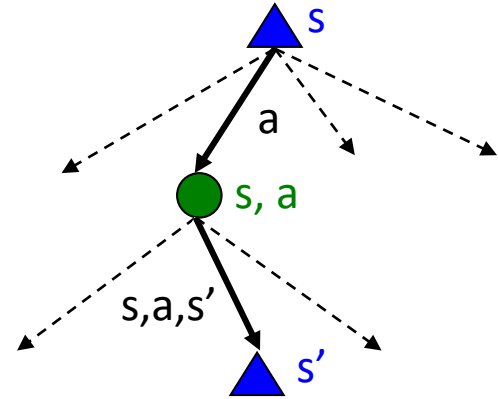
- Definition of “optimal utility” is a simple one-step lookahead relationship amongst optimal utility values

$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

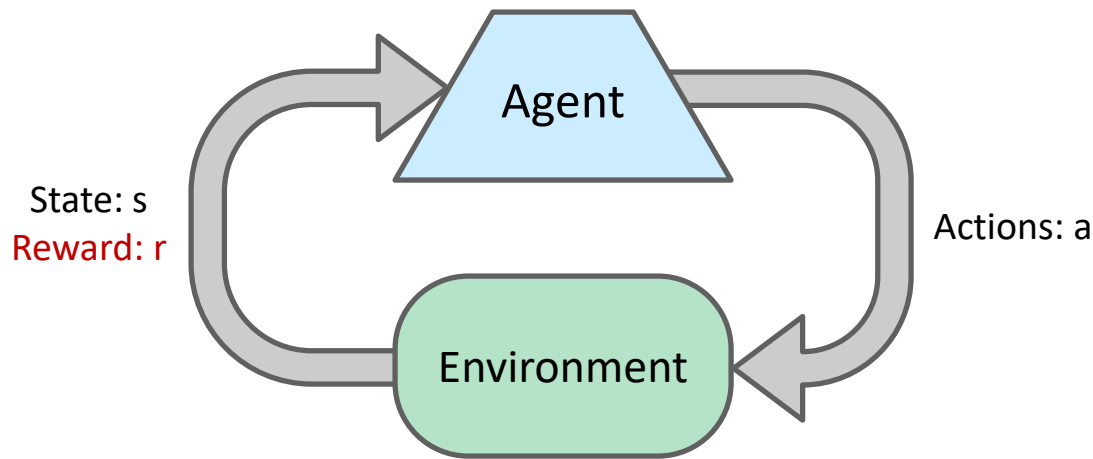
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')]$$



- These are the Bellman optimality equations, and they **characterize optimal values**

# Last Time: Reinforcement Learning (RL)



- **Basic idea:**

- Receive feedback in the form of **rewards**
- Agent's utility is defined by the reward function
- Must **learn** to act so as to **maximize expected rewards**
- All learning is based on observed samples of outcomes!

# Last Time: Reinforcement Learning (RL)

## ■ Passive vs. active

- **Both:** Dynamics of MDP unknown ---  $T(s,a,s')$  and  $R(s,a,s')$  unknown
- **Passive:** Given *fixed* policy  $\pi$ , goal is to learn state values  $V(s)$ . No choice about actions, just execute policy and learn from experience.
- **Active (what we're mostly interested in):** Agent chooses actions, with the goal of learning optimal policy  $\pi$  and maybe state values  $V(s)$ . *Exploration vs. exploitation*

## ■ Model-based (MB) vs. model-free (MF):

- **MB:** First learn MDP quantities  $T(s,a,s')$  and  $R(s,a,s')$ . Then use MDP algorithms (e.g. policy iteration) to find policy. **Pros:** Often more sample efficient. **Cons:** The “goodness” of the policy is bounded by the “goodness” of the model.
- **MF:** Do not explicitly try to learn model dynamics  $T$  and  $R$ . Instead directly learn the policy / state-values, e.g. using Q-learning. Pros/Cons are “opposite” to those of MB

# Last Time: Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

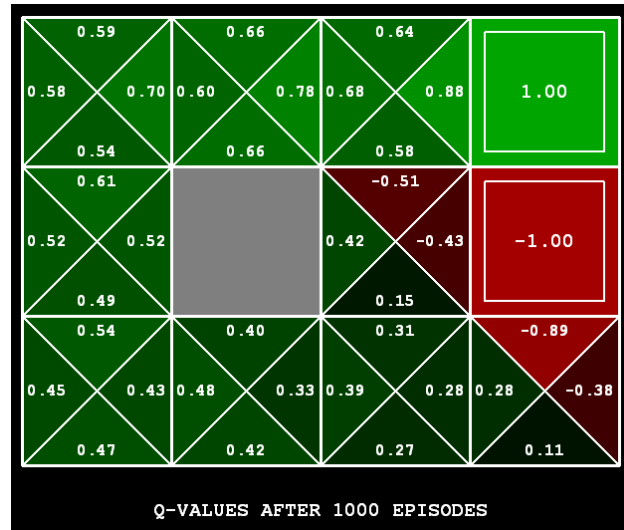
- Learn  $Q(s,a)$  values as you go

- Receive a sample  $(s, a, s', r)$
- Consider your old estimate:  $Q(s, a)$
- Consider your new sample estimate:

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

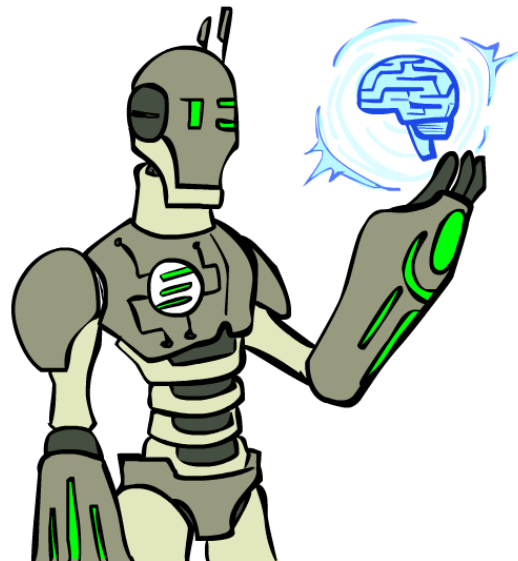
- Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) [sample]$$



# Today

- Intermediate / averaged lookahead
- Deep function approximation in RL → **deep reinforcement learning (DRL)**
- **Learning optimal policy  $\pi^*$  directly**, without intermediate step of learning  $Q^*$  or  $V^*$ , using **policy gradients**



Partial slide credits:

- Dan Klein and Pieter Abbeel, *CS188 Intro to AI* at UC Berkeley (<http://ai.berkeley.edu>);
- David Silver, *COMPM050/COMPG113 Reinforcement Learning*, UCL
- Richard Sutton and Andrew Barto, *Reinforcement Learning: An Introduction*
- Andrey Karpathy, blog post on Reinforcement Learning: <http://karpathy.github.io/2016/05/31/rl/>



# Evolution of $(s, a, r, s')$ -transitions in MDPs

- How a trajectory evolves in MDP given policy  $\pi(a \mid s)$  and dynamics  $p(r, s' \mid s, a)$
- Sample first action:  $a_0 \sim \pi(* \mid S_0 = s_0)$
- The reward  $r_1$  and next state  $s_1$  are obtained as:  $r_1, s_1 \sim p(*, * \mid S_0 = s_0, A_0 = a_0)$
- A trajectory of transitions thus evolves as:

$$s_0 \xrightarrow{\pi(*|s_0)} a_0 \xrightarrow{p(*, *|s_0, a_0)} r_1, s_1 \xrightarrow{\pi(*|s_1)} a_1 \xrightarrow{p(*, *|s_1, a_1)} r_2, s_2 \xrightarrow{\pi(*|s_2)} \dots$$
$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots)$$

- Slightly informal notation:  $\tau \sim \pi$  "trajectory sampled from policy", although in reality of course also depends on model dynamics  $p$

# Bellman Equations written as Expectations

- Last lecture we looked at the state-value Bellman equation (similar for  $Q(s,a)$ ):

$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

- This can be written more generally (and with slightly different notation, see Sutton and Barto), allowing stochastic policies and rewards:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_\pi(s')]$$

$$= \mathbb{E}_{\pi(*|s)} \left\{ \sum_{r,s'} p(r, s'|s, A) [r + \gamma v_\pi(s')] \right\}$$

$$= \mathbb{E}_{\pi(*|s)} \left\{ \mathbb{E}_{p(*, *|s, A)} \{R + \gamma v_\pi(S')\} \right\}$$

- **Remark:** We cannot change the transition probability  $p$ , only the policy  $\pi \rightarrow$  often written simply as  $v_\pi(s) = \mathbb{E}_{\pi(*|s)} \{R + \gamma v_\pi(S')\}$  or even  $v_\pi(s) = \mathbb{E}_\pi \{R + \gamma v_\pi(S')\}$

# More Notation and Definitions (Sutton and Barto)

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

***Clearer notation***

***would be:***  $v_{\pi}^t(s), q_{\pi}^t(s, a)$

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

$$s_0 \xrightarrow{\pi(*|s_0)} a_0 \xrightarrow{p(*, *|s_0, a_0)} r_1, s_1 \xrightarrow{\pi(*|s_1)} a_1 \xrightarrow{p(*, *|s_1, a_1)} r_2, s_2 \xrightarrow{\pi(*|s_2)} \dots$$

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots)$$

# Monte-Carlo (MC) Policy Evaluation

- The goal is to learn  $v_\pi$  from episodes of experience under policy  $\pi$

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

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

- The return is the total discounted reward

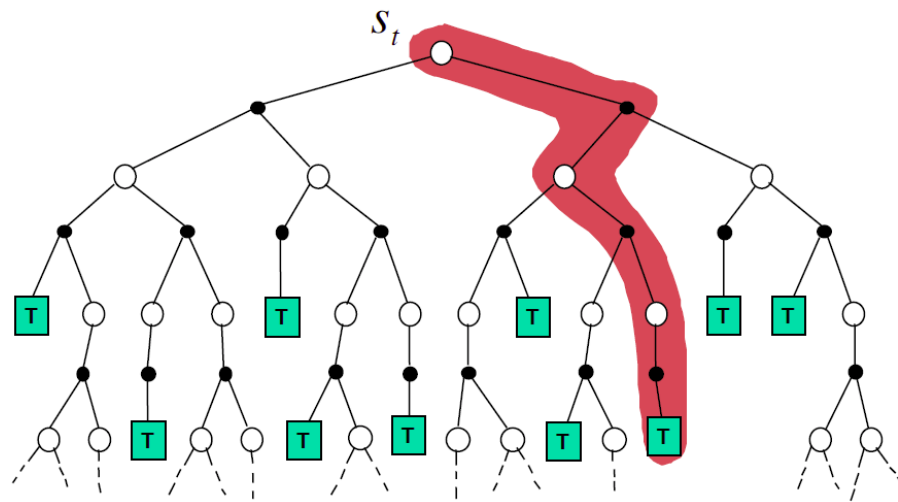
$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- The value function is the expected return

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

- **Monte-Carlo** policy evaluation uses **empirical mean return** instead of expected return

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



# Monte-Carlo (MC) Policy Evaluation Update

To evaluate state  $s$

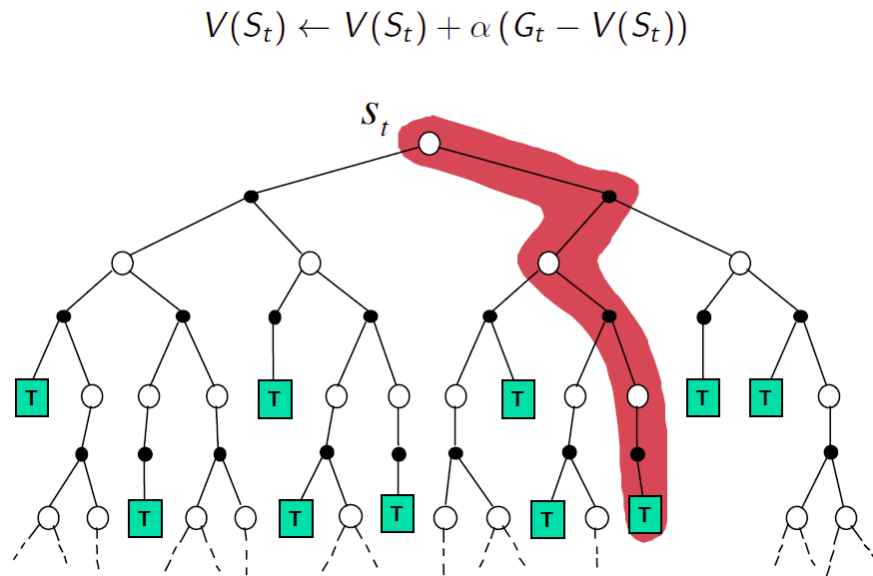
- **First / every** time-step  $t$  when state  $s$  is visited in an episode:

- Increment counter:  $N(s) \leftarrow N(s) + 1$
- Increment total return:  $S(s) \leftarrow S(s) + G_t$
- Estimate value by mean return:  $V(s) \leftarrow \frac{S(s)}{N(s)}$

- By law of large numbers

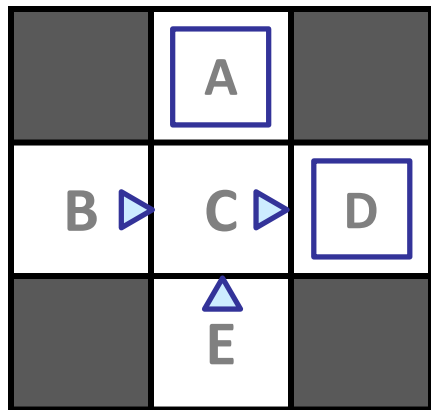
$$V(s) \rightarrow v_{\pi}(s) \text{ as } N(s) \rightarrow \infty$$

- Unbiased estimate, but high variance!



# Example: Monte-Carlo (MC) Evaluation

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

Output Values

	-10	
+8	+4	+10
	-2	

# Incremental Mean Update

---

Sample  $x$ , mean  $\mu$

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

# Incremental Monte-Carlo Updates

---

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

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

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

- In non-stationary problems, it can be useful to track a running mean, *i.e.* forget old episodes – use a learning rate:

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



# Pros and Cons of Monte-Carlo (MC) Evaluation

- **What's good about MC evaluation?**
  - Easy to understand
  - Eventually computes the correct average values, using just sample transitions
- **What bad about it?**
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

## Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

*If B and E both go to C under this policy, how can their values be different?*

# Temporal Difference (TD) Learning

Same goal as before: Learn  $v_\pi$  online from experience under policy  $\pi$

- Simplest temporal-difference learning algorithm, TD(0):

- Update  $V(S_t)$  toward estimated return  $R_{t+1} + \gamma V(S_{t+1})$

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

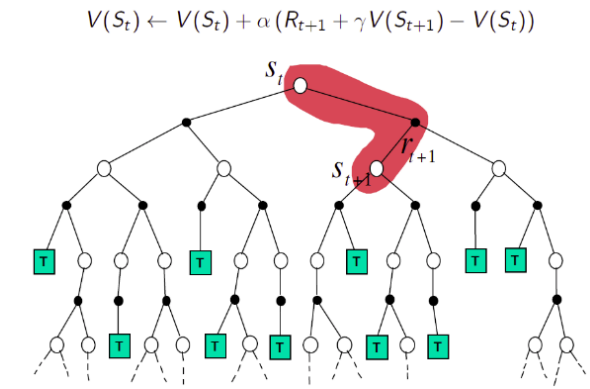
- $R_{t+1} + \gamma V(S_{t+1})$  is called the **TD target**
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  is called the **TD error**

- Compare with **Monte-Carlo, MC**:

- Update value  $V(S_t)$  toward actual return  $G_t$

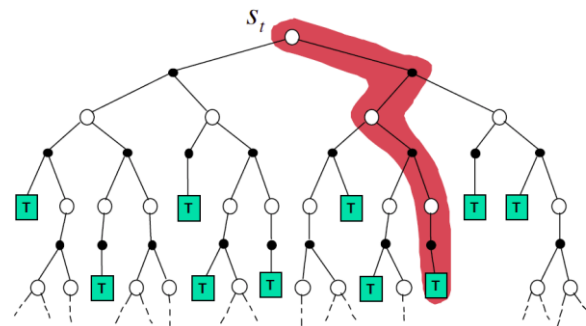
$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

- Note:** MC and TD(0) equally applicable for  $Q(s,a)$  as for  $V(s)$  – state-value function  $V$  used here for concreteness!



TD(0)  $\uparrow$  vs. MC  $\downarrow$

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



# Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 0.5$

Observed Transitions

B, east, C, -2

	0	
0	0	8
	0	

C, east, D, -2

	0	
-1	0	8
	0	

	0	
-1	3	8
	0	

$$V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + \alpha [R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

# Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

Observed Transitions

B, east, C, -2

	0	
-1	3	8
	0	

C, east, D, -2

	0	
0	3	8
	0	

	0	
0	4.5	8
	0	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

# Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

Observed Transitions

B, east, C, -2

	0	
0	4.5	8
	0	

C, east, D, -2

	0	
1.25	4.5	8
	0	

	0	
1.25	5.25	8
	0	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

# Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

Observed Transitions

E, North, C, -2

	0	
1.25	5.25	8
	0	

C, east, D, -2

	0	
1.25	5.25	8
	1.625	

	0	
1.25	5.625	8
	1.625	

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

# Bias Variance Trade-offs

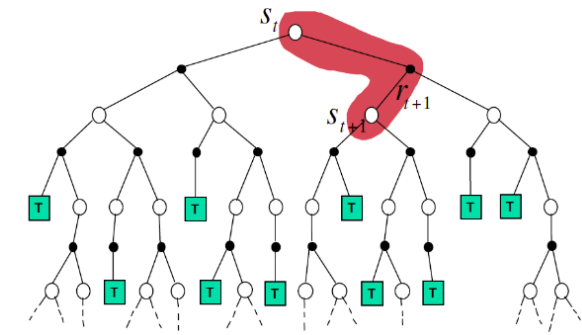
- Return  $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$  is **unbiased** estimate of  $v_\pi(S_t)$ 
  - We observe sample of **actual reward**
- True TD target  $R_{t+1} + \gamma v_\pi(S_{t+1})$  is **unbiased** estimate of  $v_\pi(S_t)$ 
  - Simply the **Bellman equation** for the state-value function
- TD target  $R_{t+1} + \gamma V(S_{t+1})$  is **biased** estimate of  $v_\pi(S_t)$ 
  - $V(S_{t+1})$  is **not actually observed** (“estimate based on an estimate”)
- TD target is much **lower variance** than the return
  - Return depends on **many** random actions, transitions, rewards
  - TD target depends on **one** random action, transition, reward

# TD vs. MC

- **TD has low variance, some bias**

- Usually more efficient than MC
- **TD(0)** converges to  $v_{\pi}(s)$ , but not always with function approximation
- Can be sensitive to initial value
- Works in non-episodic / continuing environments
- Fully exploits underlying MDP assumption

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

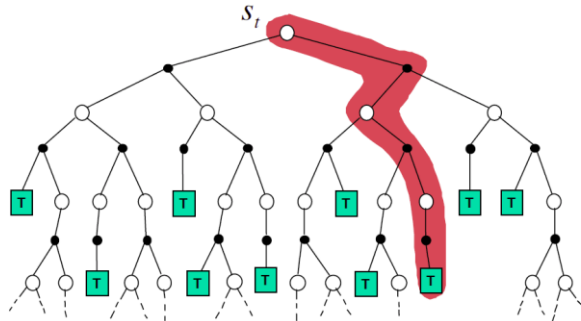


TD(0) ↑ vs. MC ↓

- **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
- Only works in episodic / terminating environments
- Does not exploit underlying MDP assumption

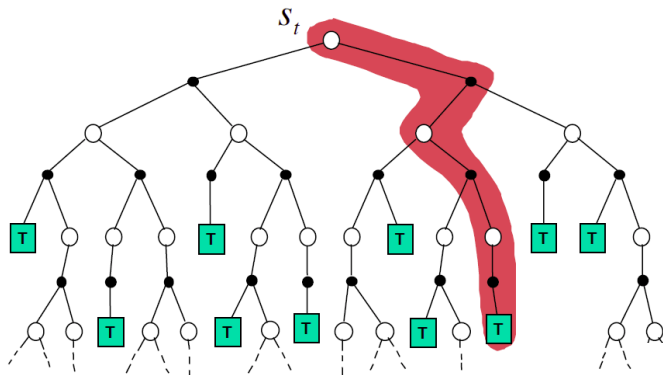
$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$





# MC vs. TD vs. DP (Dynamic Programming / Bellman)

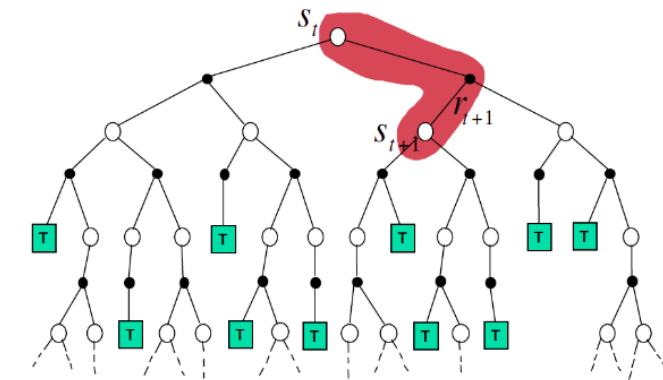
$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



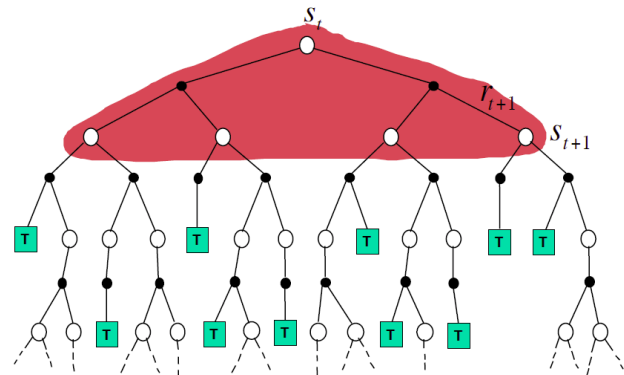
MC ↑

TD(0) ↓

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



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



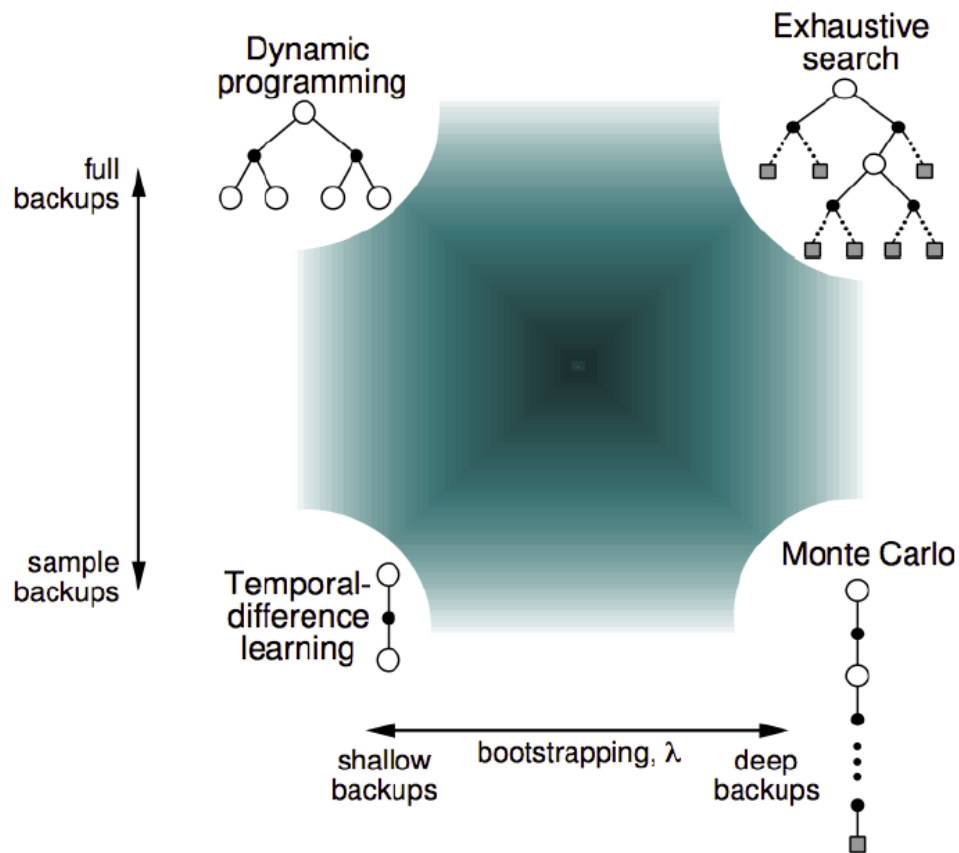
DP ↑

# Sampling vs. Bootstrapping

---

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

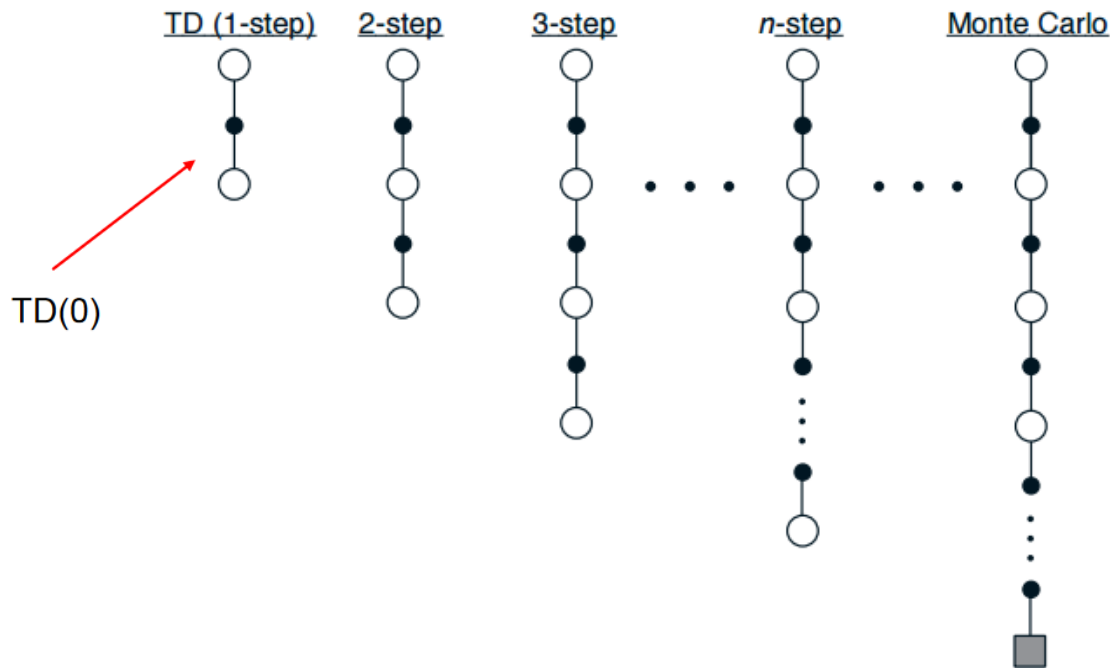
# Unified View of Reinforcement Learning



# How much lookahead when learning?

- So far (today and last lecture) we have considered only two “extremes”:
- 1) Monte-Carlo (MC) estimation (**full lookahead**):
  - We update estimates of  $V(s)$  or  $Q(s,a)$  only at the very **end of episodes**
  - Only works for **episodic / terminating environments**
  - Does not exploit underlying MDP assumption → **may be better when the system isn't truly an MDP**
  - **High variance, but unbiased**
- 2) Temporal difference learning, e.g. Q-learning (**one-step lookahead – TD(0)**):
  - Update estimate of  $V(s)$  or  $Q(s,a)$  after **every single step**
  - Update based on **immediate reward + bootstrapping** (i.e. incorporate current estimate of the values one step into the future)
  - Works in **continuing / non-terminating environments**
  - Fully exploits underlying MDP assumptions → **often better-performing when MDP assumption is true**
  - **Low variance, but can have large bias**
- 3) Today: Can we do something in between to balance the respective benefits?

# Multi-Step TD



Spectrum: **one-step backups of TD(0)  $\rightarrow$  termination-only backups of MC.** In between are the  **$n$ -step backups**, based on  **$n$  steps of real rewards and the estimated value of the  $n$ th next state**, appropriately discounted

# $n$ -Step Return

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

$$n = 1 \quad (TD) \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

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

$$\vdots \quad \vdots$$

$$n = \infty \quad (MC) \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Define the  $n$ -step return

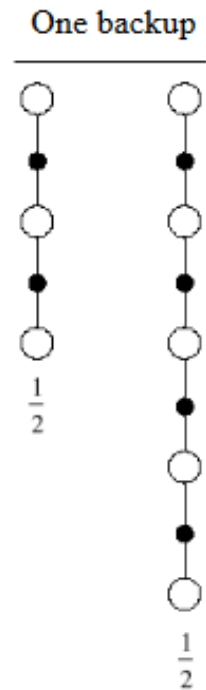
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- $n$ -step temporal-difference learning

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

# Averaging $n$ -Step Returns

- Can average  $n$ -step returns for different  $n$
- E.g. average 2-step and 4-step returns  $\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$
- A valid update not just toward any  $n$ -step return, but toward **any of their averages**
- Any set of returns can be averaged, even an infinite set, as long as the **weights on the component returns are positive and sum to 1**.

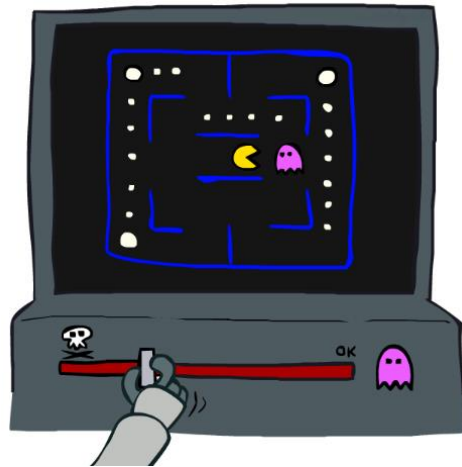


# Function Approximation for Reinforcement Learning

- Last time we looked at **tabular** Q-learning for finding action-value function  $Q(s,a)$
- Also looked at **approximate** Q-learning with **linear** function approximation

$$Q_w(s, a) = w_1 f_1(s, a) + \dots + w_n f_n(s, a)$$

- Possible also for the state-value function  $V(s)$ ,  $V_w(s)$
- **Why function approximation?**
  - Too many states and / or actions to fit in memory
  - Too slow to learn each state- / state-action value separately
  - Generalize from seen to unseen states
- We'll focus on **differentiable function approximators**
  - Linear
  - (Deep) neural networks





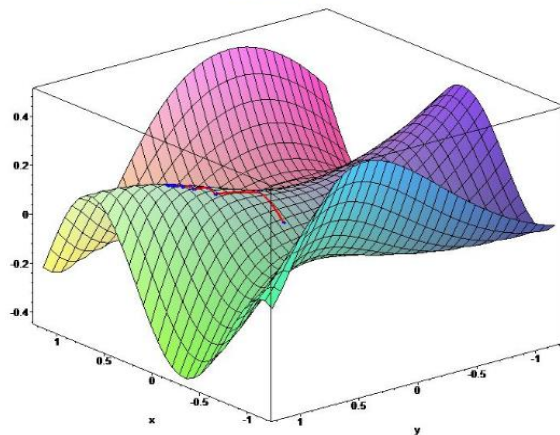
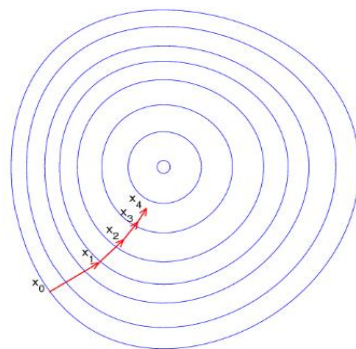
# Gradient Descent (again)

- Let  $J(\mathbf{w})$  be a differentiable function of parameter vector  $\mathbf{w}$
- The gradient of  $J(\mathbf{w})$  is:

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

- We (locally) minimize  $J(\mathbf{w})$  by adjusting  $\mathbf{w}$  in the direction of the gradient with a step-size parameter  $\alpha$ :

$$\Delta \mathbf{w} = -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$



# Value Function Approximation via Stochastic Gradient Descent (SGD)

- **Goal:** Find parameters  $\mathbf{w}$  minimizing MSE between approx. and true value functions

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

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_{\pi} [(v_{\pi}(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})] \end{aligned}$$

- ***Stochastic gradient descent samples the gradient***

$$\Delta \mathbf{w} = \alpha (v_{\pi}(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update
- However, often high variance

# What About the True Value Function $v_\pi(s)$ ?

- On previous slide, assumed true value function was given ("supervised learning")

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

- **RL is not supervised learning!** The closest thing to supervision is the **rewards**
- In practice, we use a *target* estimate of  $v_\pi(s)$

- For MC, the target is the return  $G_t$

$$\Delta \mathbf{w} = \alpha (\mathbf{G}_t - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

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

$$\Delta \mathbf{w} = \alpha (\mathbf{R}_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

# Linear Value Function Approximation

- Represent value function by a **linear combination of features**:

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^n \mathbf{x}_j(S) \mathbf{w}_j$$

- The **objective function** is then **quadratic** in the parameters  $\mathbf{w}$

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

- **SGD converges to global optimum**
- **Update rule is really simple** ("step-size  $\alpha$  pred. error  $\times$  feature value"):

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

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

# Linear Function Approximation for Q-Learning

$$\hat{Q}(s, a; \mathbf{w}) = w_1 f_1(s, a) + \cdots + w_n f_n(s, a) = \mathbf{w}^\top \mathbf{f}(s, a)$$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left\{ \left( Q(s, a) - \hat{Q}(s, a; \mathbf{w}) \right)^2 \right\} = \mathbb{E}_\pi \left\{ \left( Q(s, a) - \mathbf{w}^\top \mathbf{f}(s, a) \right)^2 \right\}$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = -2 \mathbb{E}_\pi \left\{ \left( Q(s, a) - \mathbf{w}^\top \mathbf{f}(s, a) \right) \mathbf{f}(s, a) \right\}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left( Q(s, a) - \mathbf{w}^\top \mathbf{f}(s, a) \right) \mathbf{f}(s, a)$$

- **What about  $Q(s, a)$ ?** We can **bootstrap** (just like for tabular Q-learning!)

$$Q(s, a) \approx r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left( r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}) \right) \mathbf{f}(s, a)$$

# Linear Function Approximation for Q-Learning

$$Q(s, a) \approx w_1 f_1(s, a) + \dots + w_n f_n(s, a) = \mathbf{w}^\top \mathbf{f}(s, a)$$

- Q-learning with linear Q-functions:

transition =  $(s, a, r, s')$

difference =  $\left[ r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$

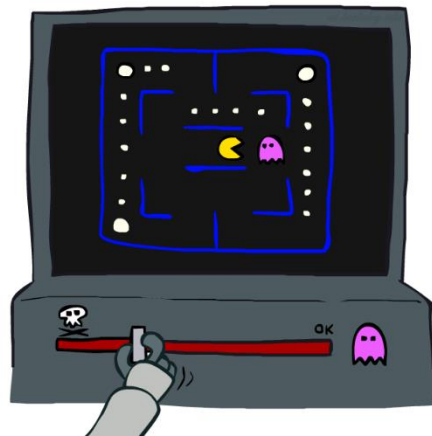
$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}]$       Exact Q's

$w_i \leftarrow w_i + \alpha [\text{difference}] f_i(s, a)$       Approximate Q's

- Intuitive interpretation:

- Adjust weights of active features

- E.g., if something unexpectedly bad happens, **blame the features that had significant magnitude:**  
disprefer all states with that state's features



# Experience Replay

- Stochastic gradient descent as described so far can be very **sample inefficient**
- **Key problem:** we are only using the very latest experience (transition) when updating our weights. Recall e.g. the linear Q-learning update rule using transition  $= (s, a, r, s')$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left( r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}) \right) \mathbf{f}(s, a)$$

- Why may it be problematic only using the very latest experience in weight update?
  - **Sample inefficient:** An experience is only used once. It can be costly to gain experiences, and would like to use them several times (similar to training for several epochs on training data in supervised learning)
  - **Non-I.I.D:** Convergence analyses (and empirical performance!) often require training samples to be i.i.d. Clearly, consecutive experiences are highly dependent!

# Stochastic Gradient Descent with Experience Replay

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

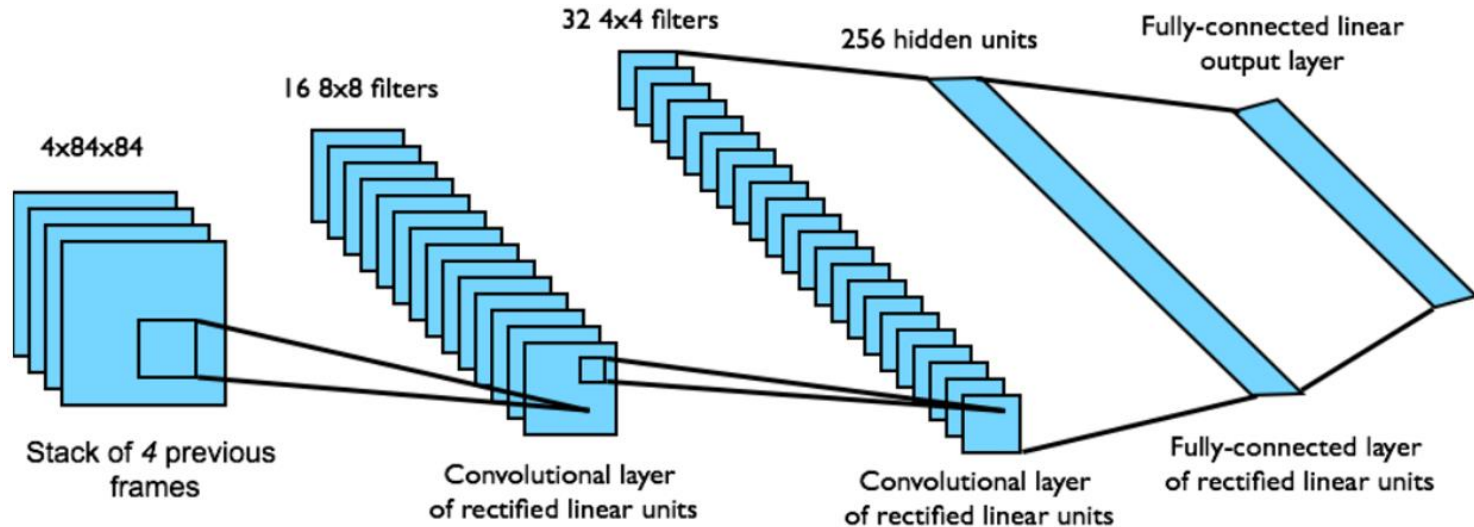
$$\mathcal{D} = \{ \langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \dots, \langle s_T, v_T^\pi \rangle \}$$

- Recall MSE-objective:  $J(\mathbf{w}) = \mathbb{E}_\pi [ (v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 ]$
- **Repeat:**
  1. Sample  $\langle \text{state}, \text{value} \rangle$  pair from experience ("replay an experience"):  $\langle s, v^\pi \rangle \sim \mathcal{D}$
  2. Apply stochastic gradient descent update:  $\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$
- Similar to supervised learning, e.g. image classification. There we have dataset of images – corresponding here to  $\mathcal{D}$  -- and we sample images, using SGD to update neural network weights. ***In RL, the learner also collects the dataset!***

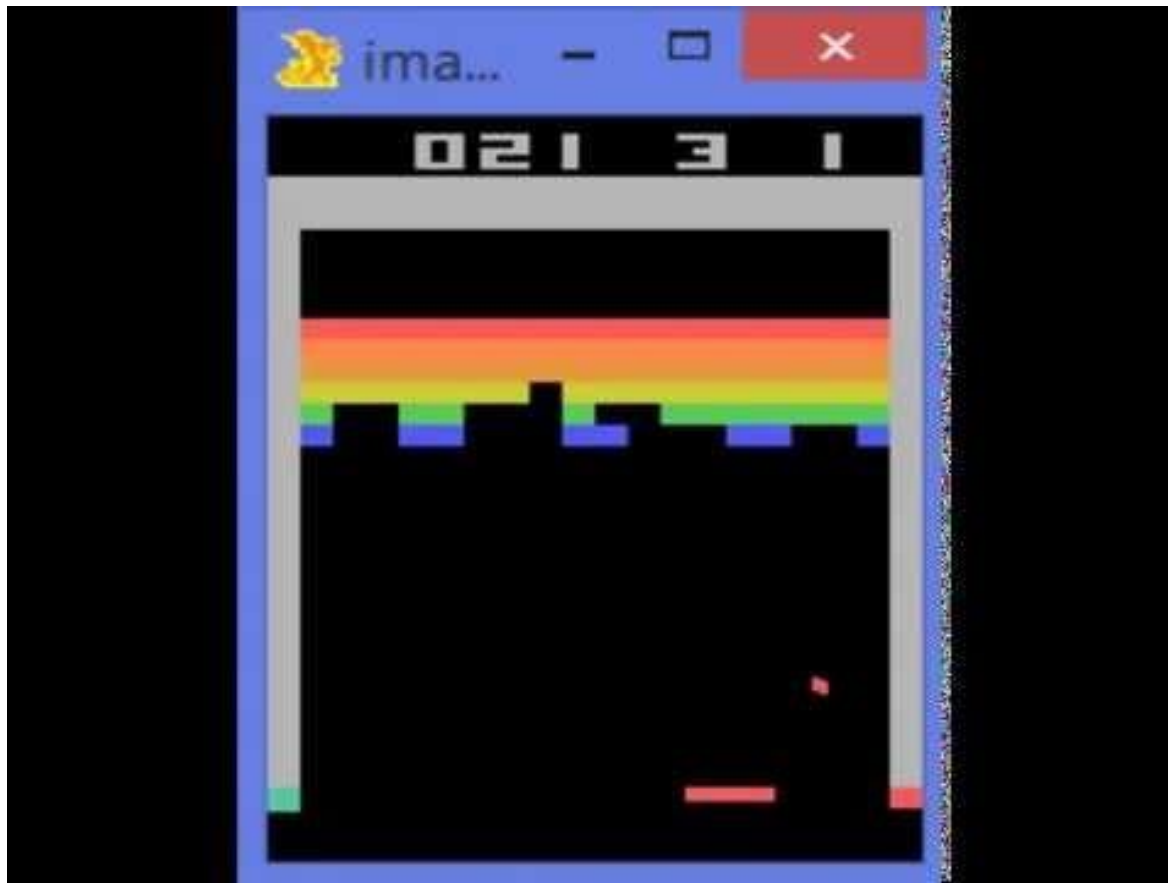


# Deep Q-Networks (DQNs)

- Use **deep neural network** instead of linear weights for approximating  $Q(s,a)$
- The fully-connected linear output layer has one channel for each action  
→ **approximation of  $Q(s,a)$  for each action  $a$  simultaneously**
- Below DQN architecture was used by Google DeepMind to achieve super-human performance on several Atari games:



# DQN for Playing Atari



# DQN – Key Concepts

- DQN uses **experience replay** and **fixed Q-targets (separate target network)**
- Take action  $a_t$  according to  $\varepsilon$ -greedy policy (see previous lecture)
- Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in the replay memory (experience buffer  $\mathcal{D}$ )
- Sample random mini-batch of transitions  $(s, a, r, s')$  from  $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters  $\mathbf{w}^-$
- Optimise MSE between Q-network and Q-learning target:

$$J(\mathbf{w}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left\{ \left( r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}^-) - \hat{Q}(s, a; \mathbf{w}) \right)^2 \right\}$$

- Every C:th step, set  $\mathbf{w}^- = \mathbf{w}$

# DQN – Full Algorithm

---

**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  $a_t = \arg \max_a Q(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in the 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 experience  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$

        Sample random minibatch of experiences  $(\phi_j, a_j, r_j, \phi_{j+1})$  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 weights  $\theta$

        Every  $C$  steps reset  $\hat{Q} = Q$

**end for**

**end for**

---

# Policy-Based Reinforcement Learning (PB-RL)

---

- Have looked at (exact / approximate) approaches for finding  $V(s)$ ,  $Q(s,a)$
- For example, from  $Q(s,a)$  a policy is obtained as  $\pi(s) = \arg \max_a Q(s,a)$
- Above paradigm for finding a policy falls into **value-based RL**
- Now we'll **directly parametrize the policy**  $\pi_{\theta}(a | s) = P(a | s; \theta)$ 
  - Use **stochastic policy to enable exploration** (instead of  $\epsilon$ -greedy approach, as for Q-learning). Also, most PB-RL algorithms assume stochastic policy to work.
- This is **policy-based RL**
- Focus will still be on **model-free reinforcement learning**

# Value-Based vs. Policy-Based RL

- **Value-based**

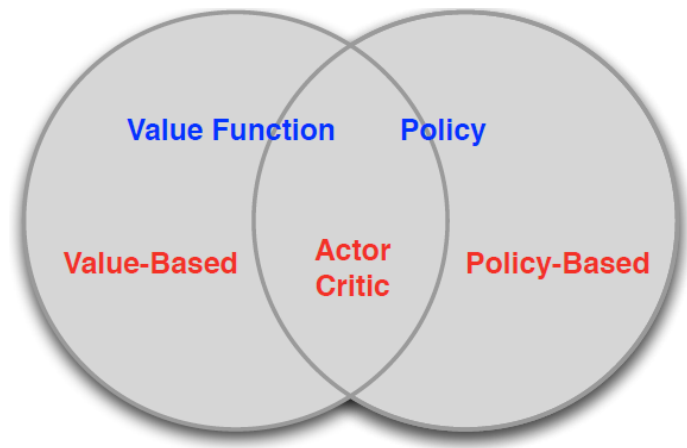
- Learn  $V(s)$  or  $Q(s,a)$
- Policy extracted from  $V(s)$  or  $Q(s,a)$

- **Policy-based**

- No intermediate learning of value functions
- Directly learn the policy  $\pi_{\theta}(a | s)$

- **Actor-Critic = Combination**

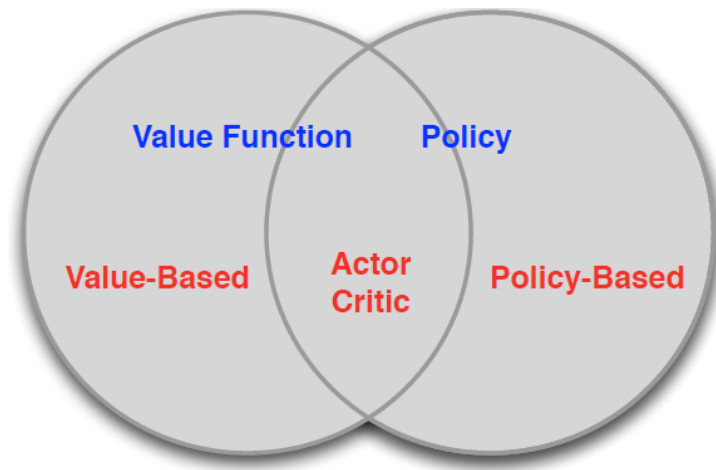
- Learn  $V(s)$  or  $Q(s,a)$  (**CRITIC**)
- Learn policy  $\pi_{\theta}(a | s)$  (**ACTOR**), often more efficiently than in pure policy-based approach



# Properties of Policy-Based RL

## ■ Advantages

- Better convergence properties
- Effective in high-dimensional or continuous action-spaces
- Can learn stochastic policies
- Actually more intuitive approach if you understand supervised learning

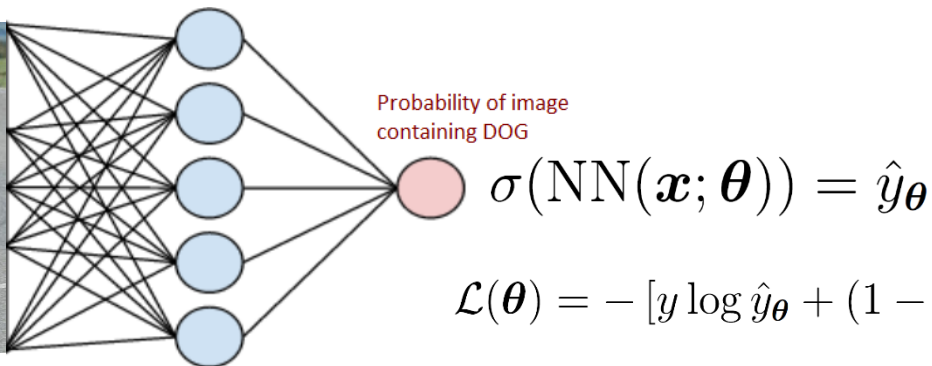


## ■ Disadvantages

- Typically converges to a local instead of global optimum (however also true for value-based RL with function approximation)
- Evaluating policy (expected reward) is typically inefficient and high variance

# Recall: Supervised Learning (in Deep Networks)

- "Does the below image contain a dog?" Answer:  $y = 0$  -- *no dog!*
- The "decision" if it contains a dog is **soft**, i.e. a probability from **differentiable** neural network layers (we can make small changes in the parameters to obtain small changes in the output prediction – no hard decisions needed in training!)
- Also, no actor (agent) affecting "next states" and "returns" (losses)



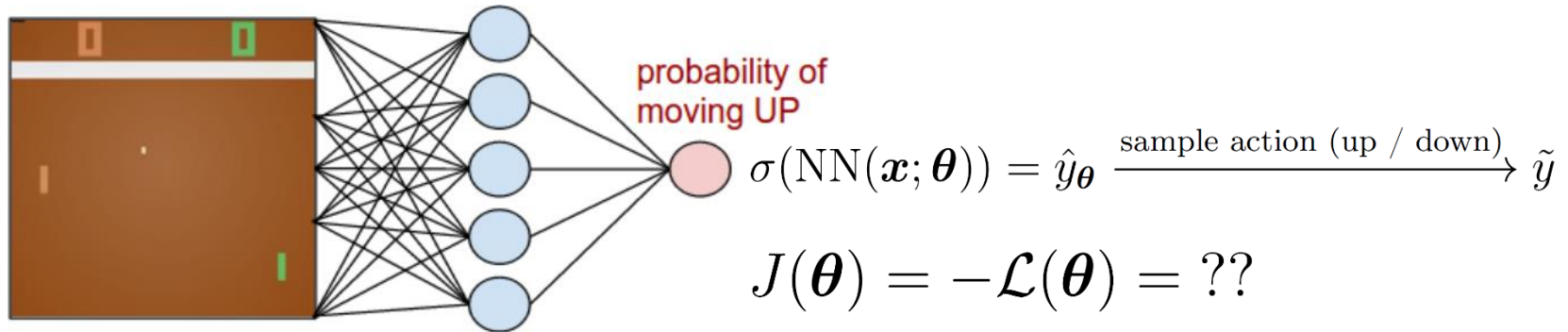
$$\mathcal{L}(\boldsymbol{\theta}) = -[y \log \hat{y}_{\boldsymbol{\theta}} + (1 - y) \log(1 - \hat{y}_{\boldsymbol{\theta}})]$$

$$J(\boldsymbol{\theta}) = -\mathcal{L}(\boldsymbol{\theta}) = [y \log \hat{y}_{\boldsymbol{\theta}} + (1 - y) \log(1 - \hat{y}_{\boldsymbol{\theta}})] \cdot (+1)$$



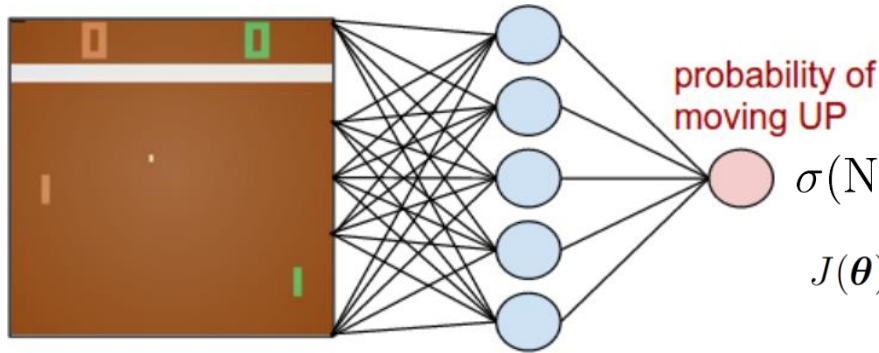
# Intuition of Policy-Based RL

- “Should I move up, to increase chances of winning?” Ans: How should I know?!
- Above question may be difficult to answer in supervised learning framework
  - What would the loss function be? What about training data?
  - Can use (possibly very delayed) reward, such as +1 for winning -1 for losing the game
- Note that the *probability* of moving up can be produced via differentiable neural network layers, but a **hard decision** must be made both in training and testing, typically by sampling ( $\sim$ ) from the action probability distribution



# Intuition of Policy-Based RL

- "Should I move up, to increase chances of winning?"
- **Policy-based RL solution:** Wait and see! Use sampled action  $\tilde{y}$  as a "fake label", since we don't have "real labels" as for image classification
- One more thing: Multiply cross-entropy with the observed future cumulative reward  $\rightarrow$  will increase probability of good actions, and decrease for bad actions



$$\sigma(\text{NN}(\mathbf{x}; \boldsymbol{\theta})) = \hat{y}_{\boldsymbol{\theta}} \xrightarrow{\text{sample action (up / down)}} \tilde{y}$$
$$J(\boldsymbol{\theta}) = [\tilde{y} \log \hat{y}_{\boldsymbol{\theta}} + (1 - \tilde{y}) \log(1 - \hat{y}_{\boldsymbol{\theta}})] \cdot \left( \sum_j r_j \right)$$

**Recall corresponding for supervised learning:**

$$J(\boldsymbol{\theta}) = -\mathcal{L}(\boldsymbol{\theta}) = [y \log \hat{y}_{\boldsymbol{\theta}} + (1 - y) \log(1 - \hat{y}_{\boldsymbol{\theta}})] \cdot (+1)$$

# Objective function for policy-based RL

- **Goal:** Find best parameters  $\theta$  for stochastic policy  $\pi_{\theta}(a | s)$

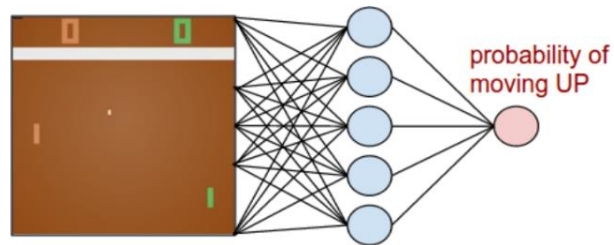
$$s_0 \xrightarrow{\pi_{\theta}(*|s_0)} a_0 \xrightarrow{p(*, *|s_0, a_0)} r_1, s_1 \xrightarrow{\pi_{\theta}(*|s_1)} a_1 \xrightarrow{p(*, *|s_1, a_1)} r_2, s_2 \xrightarrow{\pi_{\theta}(*|s_2)} \dots a_{T-1} \xrightarrow{p(*, *|s_{T-1}, a_{T-1})} r_T, s_T$$

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots, a_{T-1}, r_T, s_T)$$

- Assume:  $\pi_{\theta}(a | s)$  stochastic and differentiable when nonzero

- **Key question:** How to measure quality of  $\pi_{\theta}$ ?

- **Proposal:**  $J(\theta) = \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\}$



# Maximizing Proposed Objective Function

- **Objective function:** Expected (discounted) cumulative return from initial state by following policy  $\pi_{\theta}(a | s)$

$$J(\theta) = \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\}$$

- As usual, use the gradient as search direction: **gradient ascent!**

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\}$$

- **Problem:** That gradient looks complicated! What to do?
- **Solution:** We'll use some "tricks" introduced on the next slides

# Trick 1: The Log-Derivative Trick

---

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathbb{E}_{f_{\boldsymbol{\theta}}} \{g(x)\} &= \nabla_{\boldsymbol{\theta}} \int f_{\boldsymbol{\theta}}(x) g(x) dx \\ &= \int \frac{f_{\boldsymbol{\theta}}(x)}{f_{\boldsymbol{\theta}}(x)} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(x) g(x) dx \\ &= \int f_{\boldsymbol{\theta}}(x) \nabla_{\boldsymbol{\theta}} \log f_{\boldsymbol{\theta}}(x) g(x) dx \\ &= \mathbb{E}_{f_{\boldsymbol{\theta}}} \{ \nabla_{\boldsymbol{\theta}} \log f_{\boldsymbol{\theta}}(x) g(x) \}\end{aligned}$$

- **Great!** The gradient landed *inside* the expectation! (Enables sampling)

# Trick 2: Getting rid of those MDP dynamics

$$s_0 \xrightarrow{\pi_{\theta}(*|s_0)} a_0 \xrightarrow{p(*, *|s_0, a_0)} r_1, s_1 \xrightarrow{\pi_{\theta}(*|s_1)} a_1 \xrightarrow{p(*, *|s_1, a_1)} r_2, s_2 \xrightarrow{\pi_{\theta}(*|s_2)} \dots a_{T-1} \xrightarrow{p(*, *|s_{T-1}, a_{T-1})} r_T, s_T$$

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots, a_{T-1}, r_T, s_T)$$

$$\begin{aligned}\nabla_{\theta} \log \pi_{\theta}(\tau) &= \nabla_{\theta} \log \left( \mu(s_0) \prod_{t=0}^{T-1} \pi_{\theta}(a_t|s_t) p(r_{t+1}, s_{t+1}|s_t, a_t) \right) \\ &= \nabla_{\theta} \left( \log \mu(s_0) + \sum_{t=0}^{T-1} [\log \pi_{\theta}(a_t|s_t) + \log p(r_{t+1}, s_{t+1}|s_t, a_t)] \right) \\ &= \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t) = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t)\end{aligned}$$

- **Great!** We don't need to know / estimate the MDP dynamics!

# Putting it all together → Policy Gradient

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\}$$

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{f_{\boldsymbol{\theta}}} \{g(x)\} = \mathbb{E}_{f_{\boldsymbol{\theta}}} \{ \nabla_{\boldsymbol{\theta}} \log f_{\boldsymbol{\theta}}(x) g(x) \}$$

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) = \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t)$$

- We finally arrive at a practical form for the policy gradient:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left\{ \underbrace{\left( \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t) \right)}_{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau)} \underbrace{\left( \sum_{t=0}^{T-1} \gamma^t R_{t+1} \right)}_{g(\tau)} \middle| S_0 = s_0 \right\}$$

# Past Rewards Unaffected by Future Actions

$$\begin{aligned}\nabla_{\theta} \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\} &= \mathbb{E}_{\pi_{\theta}} \left\{ \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \left( \sum_{t=0}^{T-1} \gamma^t R_{t+1} \right) \middle| S_0 = s_0 \right\} \\ &= \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left( \sum_{t'=0}^{T-1} \gamma^{t'} R_{t'+1} \right) \right) \middle| S_0 = s_0 \right\} \\ &= \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left( \sum_{t'=t}^{T-1} \gamma^{t'} R_{t'+1} \right) \right) \middle| S_0 = s_0 \right\}\end{aligned}$$

- Intuition of last equality: **Past rewards are unaffected by future actions!** When assessing action  $a_t \sim \pi_{\theta}(* | s_t)$  it makes sense to only account for what happens **onwards in time!**



# Sample-Based Policy Gradient

$$\begin{aligned}\nabla_{\theta} \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\} &= \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left( \sum_{t'=t}^{T-1} \gamma^{t'} R_{t'+1} \right) \right) \middle| S_0 = s_0 \right\} \\ &\approx \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \left( \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'+1}^i \right) \right)\end{aligned}$$

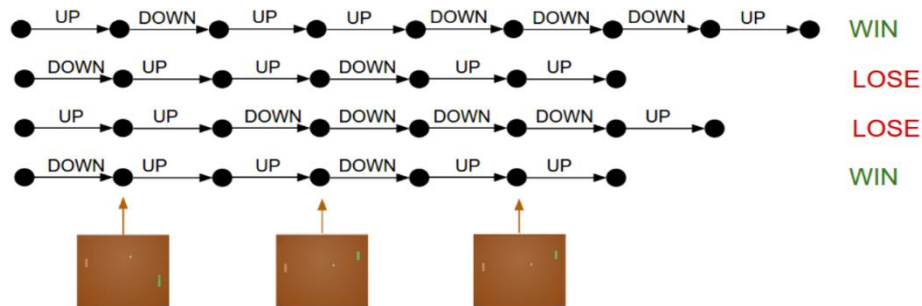
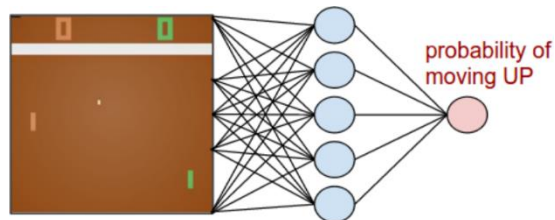
- Above: **Monte-Carlo estimate** of policy gradient (unbiased, noisy)
- Analogous to TD-learning, could use other way to estimate future discounted reward. In fact, the **policy gradient theorem** states:

$$\nabla_{\theta} \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\} = \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q^{\pi_{\theta}}(s_t, a_t) \middle| S_0 = s_0 \right\}$$

# Interpreting the policy gradient

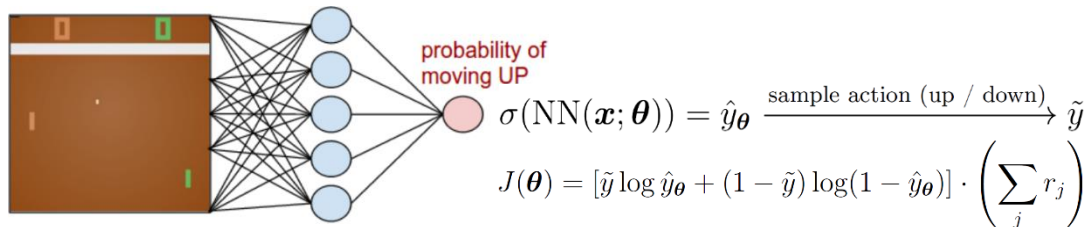
$$\nabla_{\theta} \mathbb{E}_{\pi_{\theta}} \left\{ \sum_{t=0}^{T-1} \gamma^t R_{t+1} \middle| S_0 = s_0 \right\} \approx \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T-1} \left( \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \left( \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'+1}^i \right) \right)$$

- Run  $M$  trajectories given current policy  $\pi_{\theta}(a | s)$ . When taking action  $a_t^i \sim \pi_{\theta}(* | s_t^i)$  in  $i$ :th trajectory, see how it goes (onwards in that trajectory). Update  $\theta$  to increase (log-)probability of actions yielding positive future cumulative return, and decrease otherwise



# Relating derived policy gradient to initial example

- "Should I move up, to increase chances of winning?"



- It holds that  $\log \hat{y}_{\boldsymbol{\theta}} = \log \sigma(\text{NN}(\mathbf{x}; \boldsymbol{\theta})) = \log\text{-prob. of action "move up" given current frame } \mathbf{x}$   
 $= \log \pi_{\boldsymbol{\theta}}(a_t = 1 | s_t)$  ( $\mathbf{x}$  is the state  $s_t$  here)
- Assume that  $\tilde{y} = 1$  (sampled action: "move up", where  $\tilde{y}$  is the action  $a_t$  here)
- Then  $J(\boldsymbol{\theta}) = [1 \cdot \log \hat{y}_{\boldsymbol{\theta}} + (1 - 1) \cdot (1 - \log \hat{y}_{\boldsymbol{\theta}})] \cdot \left( \sum_j r_j \right) = \log \pi_{\boldsymbol{\theta}}(a_t = 1 | s_t) \cdot \left( \sum_{t'=t}^{T-1} \gamma^{t'} R_{t'+1} \right)$
- With gradient  $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t = 1 | s_t) \cdot \left( \sum_{t'=t}^{T-1} \gamma^{t'} R_{t'+1} \right)$
- Compare to prev. slides (full objective of course is average over all individual actions within individual trajectories!):  

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{T-1} \left( \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^i | s_t^i) \left( \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'+1}^i \right) \right)$$

# Reading

---

*Reinforcement Learning: An Introduction (2<sup>nd</sup> edition), Sutton & Barto 2016*

- **Chapter 7.1**
  - n-step TD Prediction
- **Chapter 9.1 – 9.4, 9.6**
  - Value Function Approximation
  - Stochastic Gradient Descent
  - Artificial Neural Networks
- **Chapter 13.1 – 13.3**
  - Policy Gradient Methods