Deep Q Learning: From Paper to Code

Temporal Difference Learning

Last Time ...

- Explore for long term or exploit for short term?
- Many solutions → epsilon greedy
- Occasionally random, occasionally greedy

Refining Value Function Estimate

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

Estimate of value of a policy

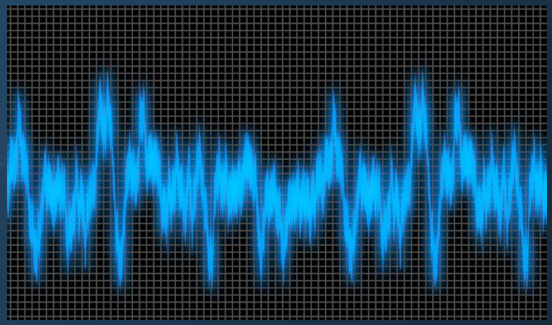
Refine estimate

New estimate = old estimate + step size (target – old estimate)

Target?

New estimate = old estimate + step size ($\boxed{\text{target}}$ – old estimate)

Direction of update

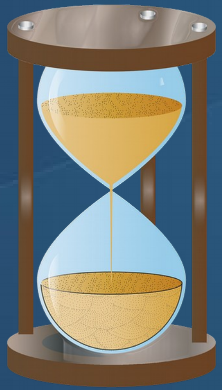


May be noisy

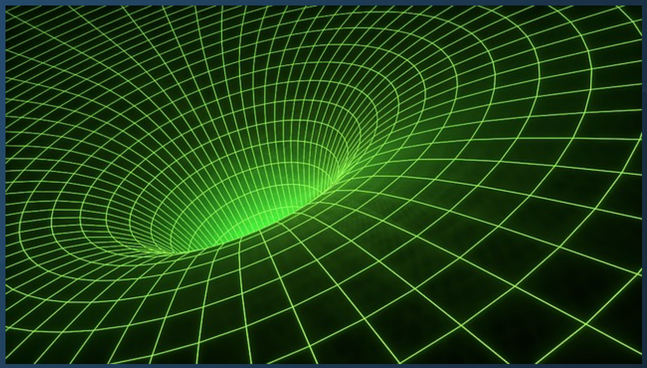
R_t

Step Size?

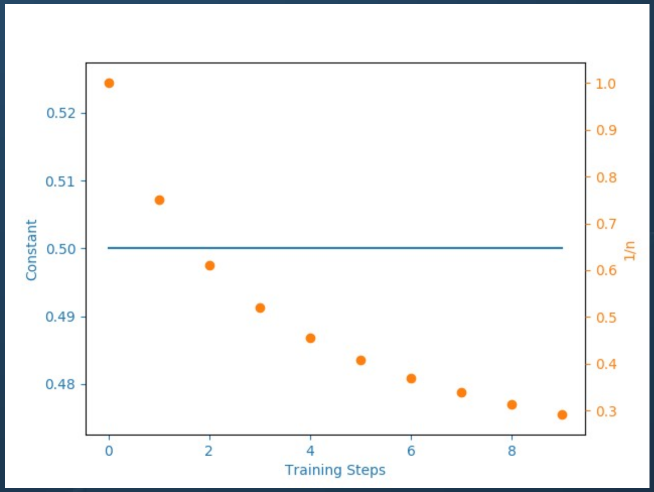
New estimate = old estimate + $\boxed{\text{step size}}$ (target – old estimate)



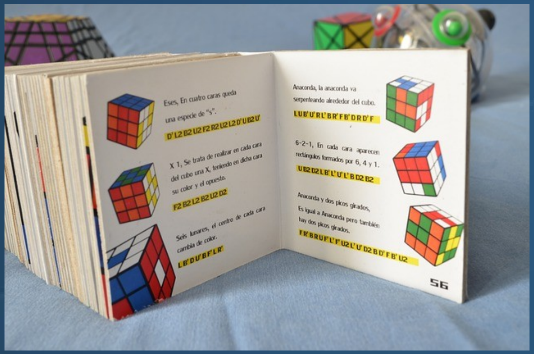
Control rate of change



Many forms



Update Frequency



Algorithm dependent

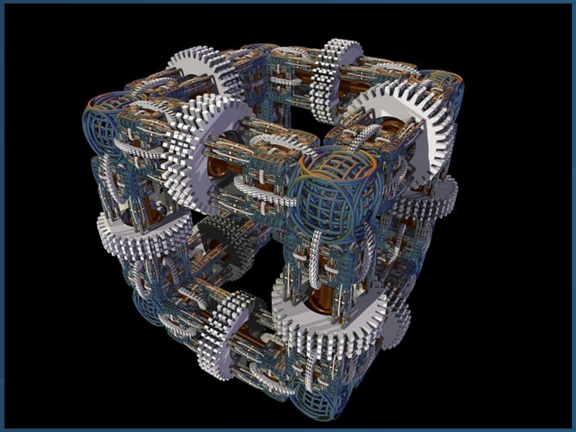
Monte Carlo → end of episode

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

Sum of discounted rewards that follow current time

New estimate = old estimate + step size (target – old estimate)

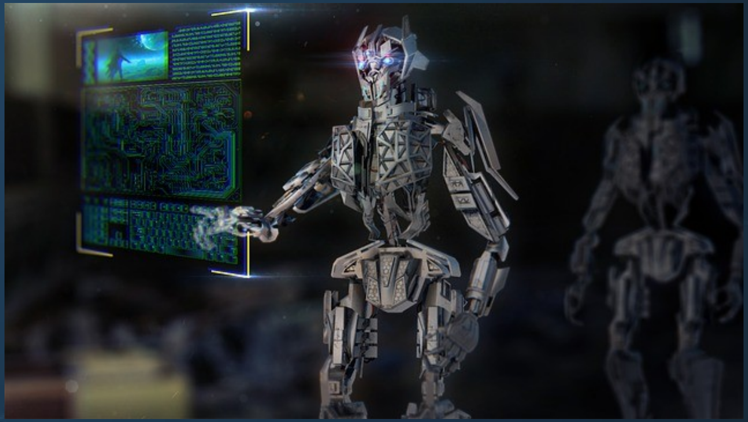
Temporal Difference Learning



Q learning

Update at each time step

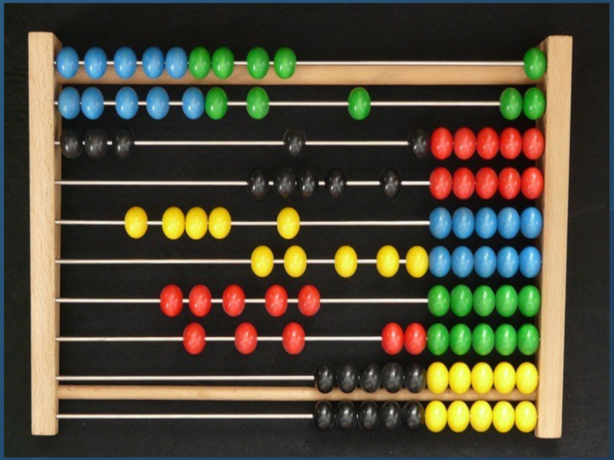
Online learning



Non episodic tasks

Temporal Difference Updates

$$\boxed{V(s_t) = V(s_t) + \alpha\left(R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right)}$$

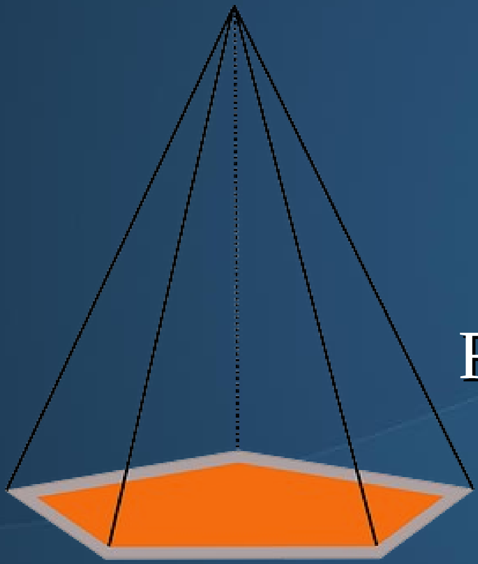


Using earlier estimate to update

<u>Bootstrapping</u>

Importance of initial estimate?

On Convergence



Fixed Π and small α → converge with good coverage

Hundreds of visits or so

How to use in Q learning?

Application to Q Learning

Have to consider action-value function!

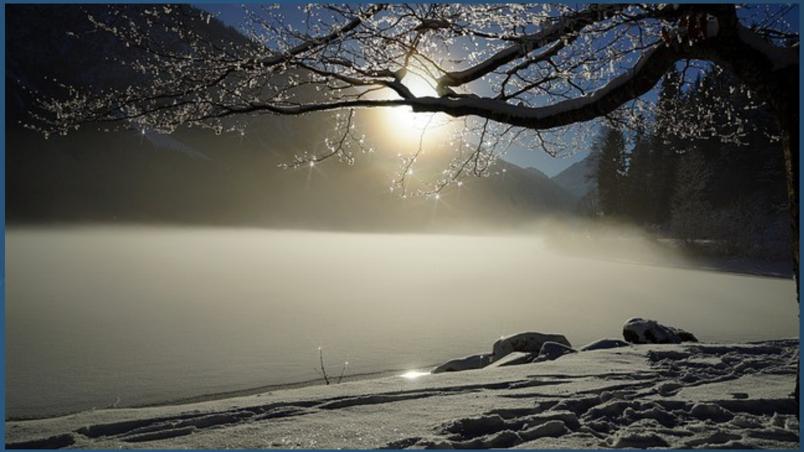
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left(R_{t+1} + \gamma \max_a Q(s_{t+1}, a_{max}) - Q(s_t, a_t)\right)$$

Compare to

$$V(s_t) = V(s_t) + \alpha\left(R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right)$$

Use maximal Q for update equation

In the Frozen Lake



Small number of states, actions

	Action 1	Action 2	Action n
State 1	Q(1,1)	Q(1,2)	Q(1,n)
State 2	Q(2,1)	Q(2,2)	Q(2,n)
State m	Q(m,1)	Q(m,2)	Q(m,n)

Tabular Learning Method

Implementation Details

Epsilon-greedy action selection

For a given state (s) look up Q for each action (a)

Max action or random action

Use reward to update Q for s, a

Dont’ reset table → coverage of all states

Off Policy Learning

Epsilon greedy to update greedy

Off Policy Learning

SARSA → On policy

Check out my YouTube for more

Q Learning Algorithm

Initialize Q for all states s and actions a

Initialize $\alpha = 0.001$ $\gamma = 0.9$ $\epsilon_{max} = 1.0, \epsilon_{min} = 0.01$

Repeat for n_episodes

Initialize state s

For each step of episode

Choose a with epsilon-greedy

Perform a, get new state s’ and reward r

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a} Q(s', a_{max}) - Q(s,a)\right)$$

s = s’

Agent should be a class

Q is a dictionary

Decrement epsilon over time

Separate files

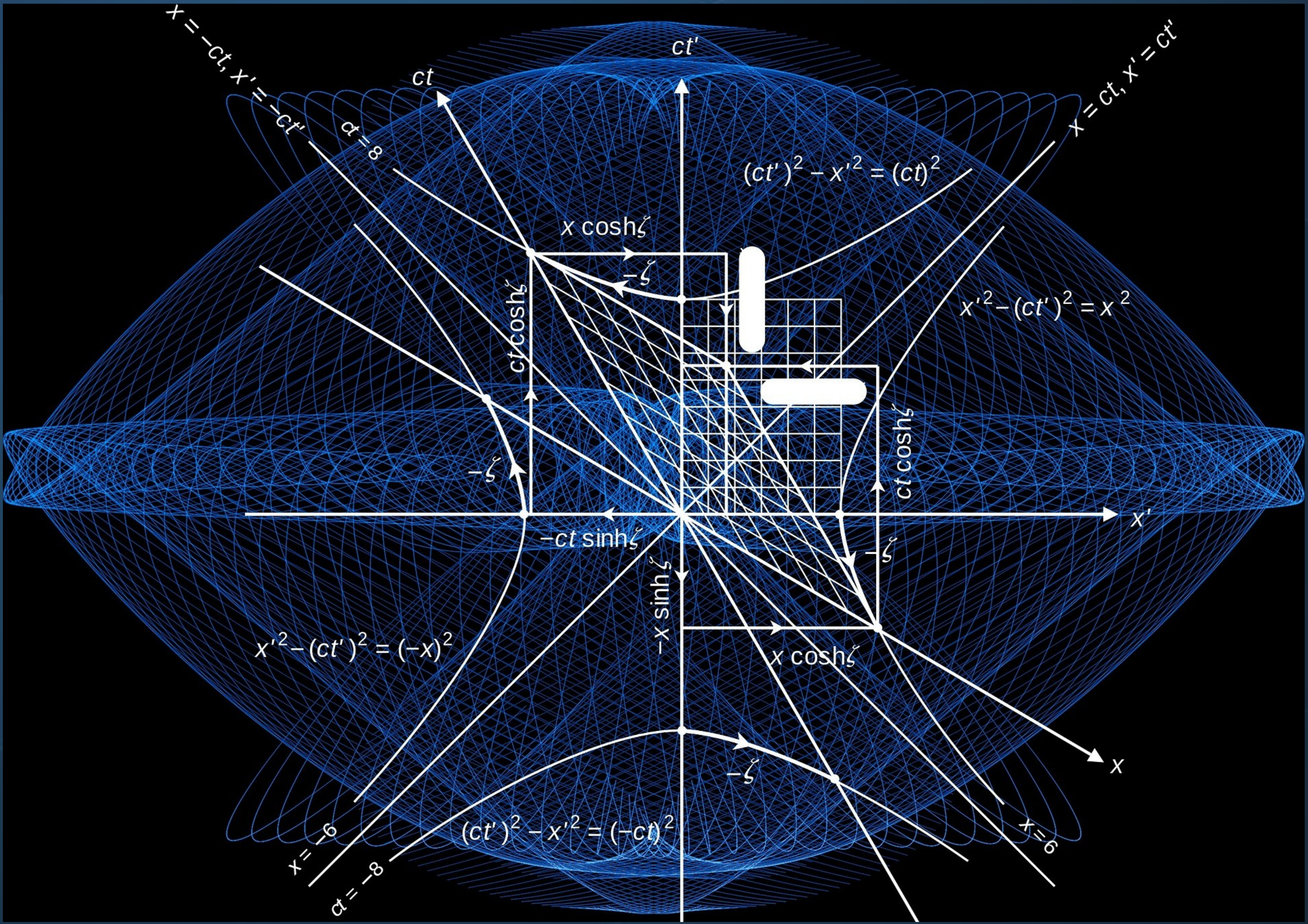
Plot average score over 100 games

500,000 total games

Summary

- Temporal difference learning is online
- Bootstrap learning
- Q learning is tabular and off policy
- Achieve 70% win rate, up from 20%

Up Next



x = −ct, x′ = −ct′
ct
ct′
x = ct, x′ = ct′
ct = 8
(ct′)² − x′² = (ct)²
x coshζ
−ζ
ct coshζ
x′² − (ct′)² = x²
−ct sinhζ
x′
−x sinhζ
x′² − (ct′)² = (−x)²
x coshζ
x
(ct′)² − x′² = (−ct)²
x = −6
ct = −8
x = 6