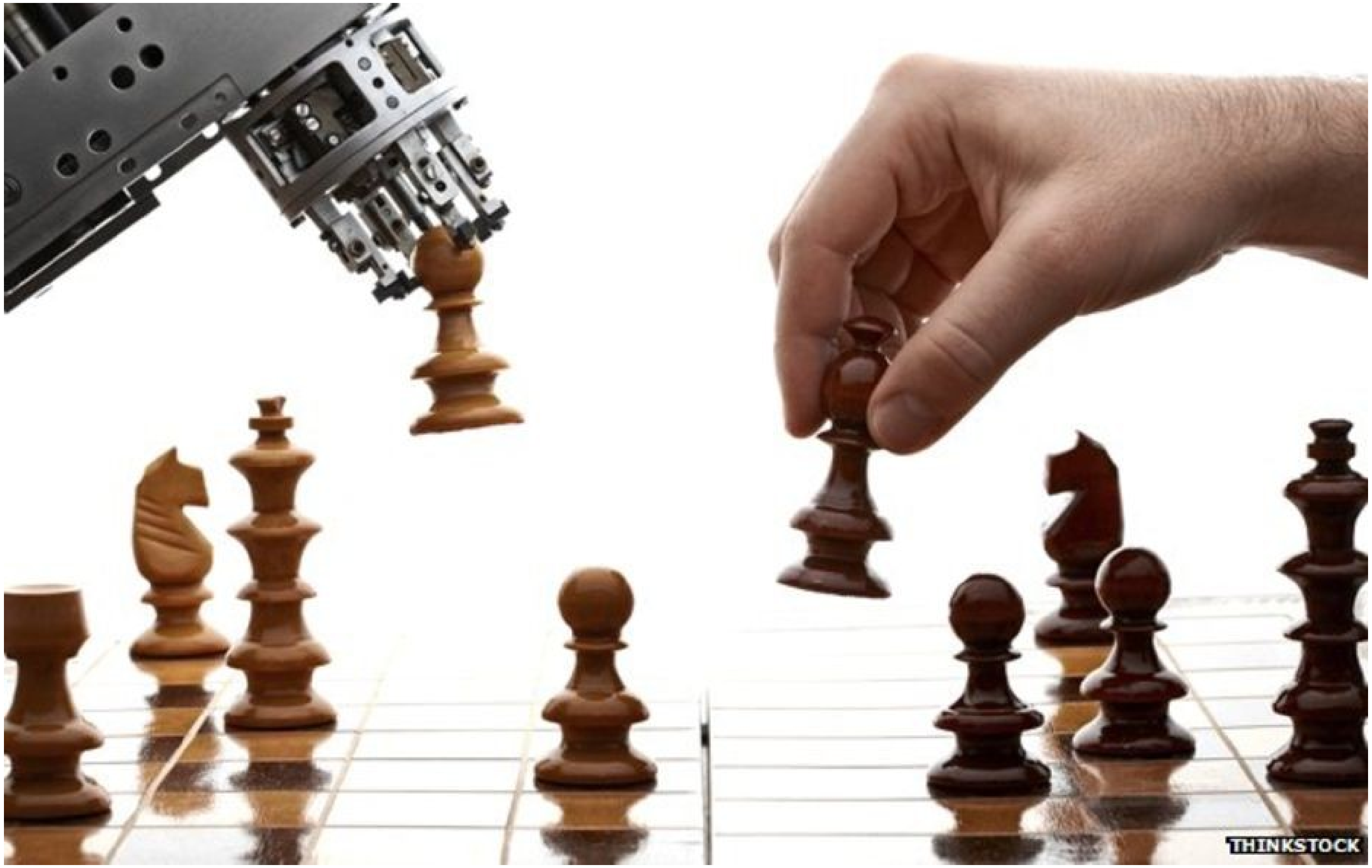
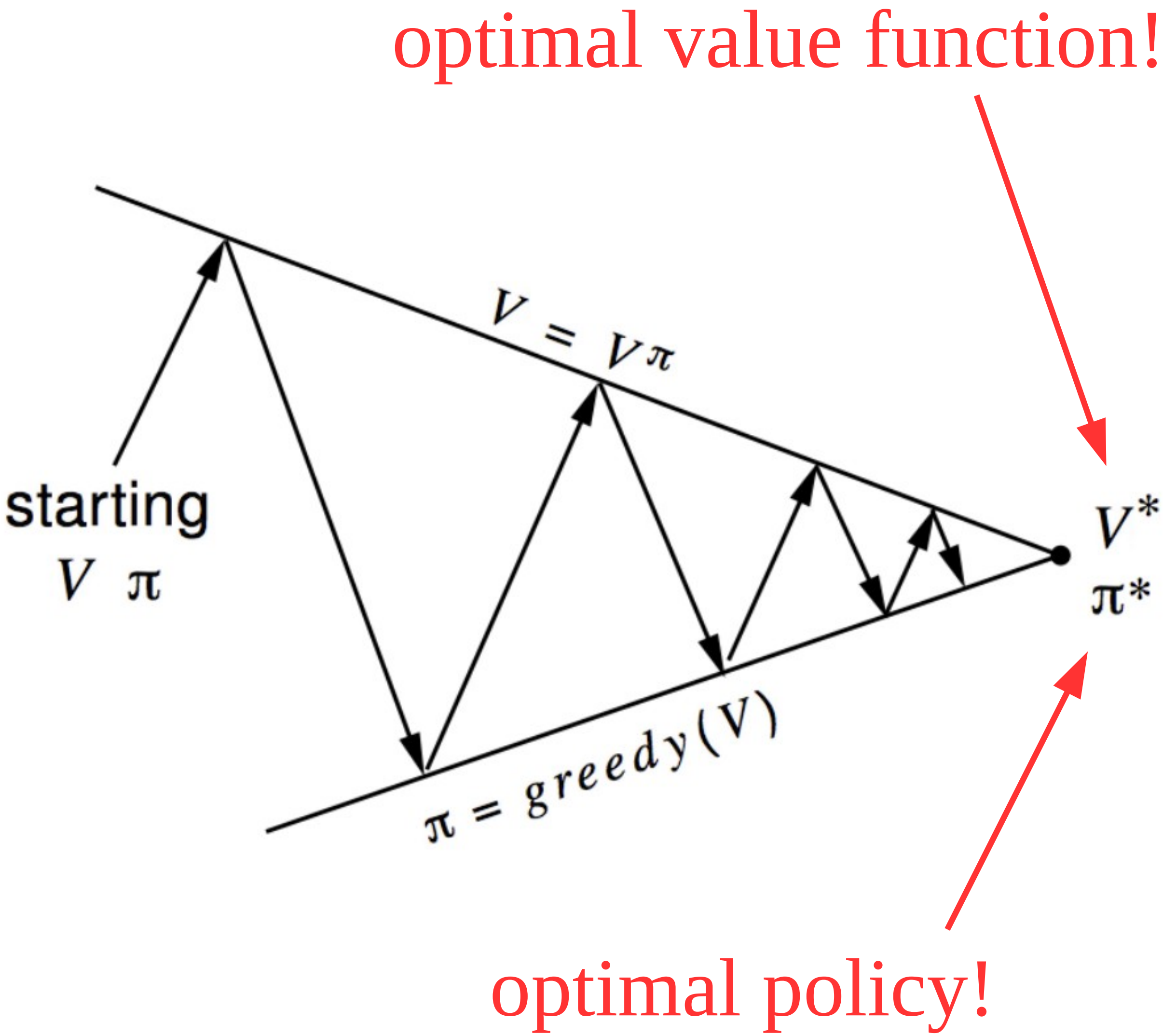
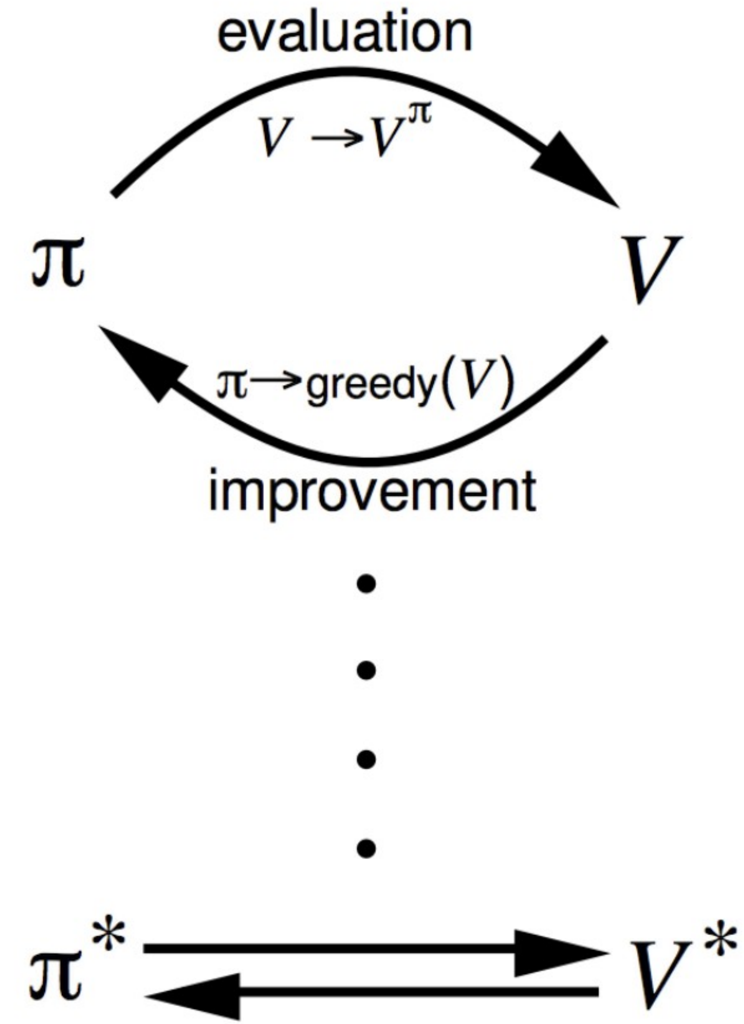


Introduction to Artificial Intelligence
COSC 4550 / COSC 5550

Professor Cheney
10/9/17



optimal value function!
$V = V\pi$
starting
V π
V^*
π^*
$\pi = greedy(V)$
optimal policy!



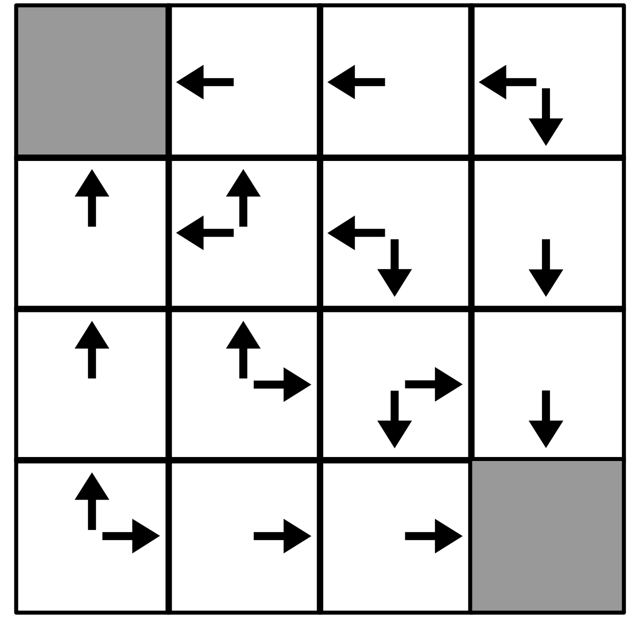
evaluation
$V \rightarrow V^\pi$
π
V
$\pi \rightarrow \text{greedy}(V)$
improvement
π^*
V^*

current value (V_k) for a random policy

greedy policy (π_k) for a this value function

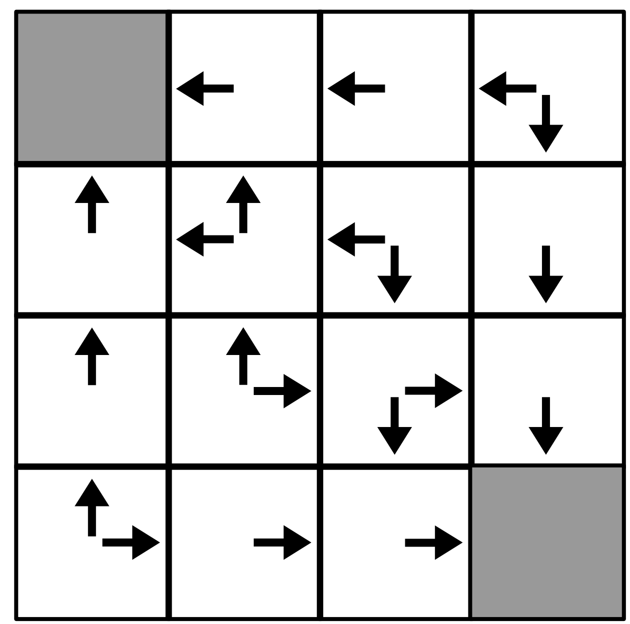
k=3

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



k=∞

0.0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0.0



value iteration “backs-up” reward
information from the goal/terminal state

it took us as many iterations of these back-ups
as there were steps from the furthest state to the goal

but many problems aren't “episodic”
and have no distinct terminal state,
and we still want to be able to maximize
reward in those settings

how can we handle infinitely long episodes?

we could just set an arbitrary cut-off time
(make in infinite “horizon” problem into a finite horizon)

$$V = \Sigma_{t=0:h} R(s_t)$$

this is a nice and simple approach,
but it introduces a problem:

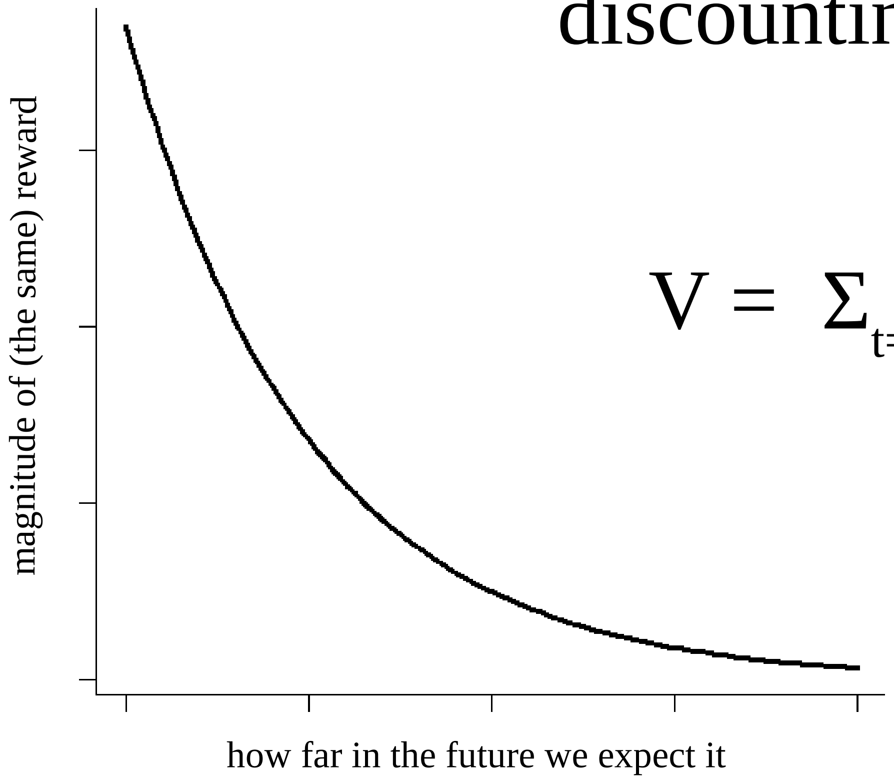
the value of a given state may differ
based on how many timesteps are left!

this means that we are asking the agent to
learn a “nonstationary” function
(which is very difficult!)

most algorithms are build on the assumption of
independent and identically distributed random variables
(i.i.d.)

how else could we handle infinitely long episodes?

discounting over time!



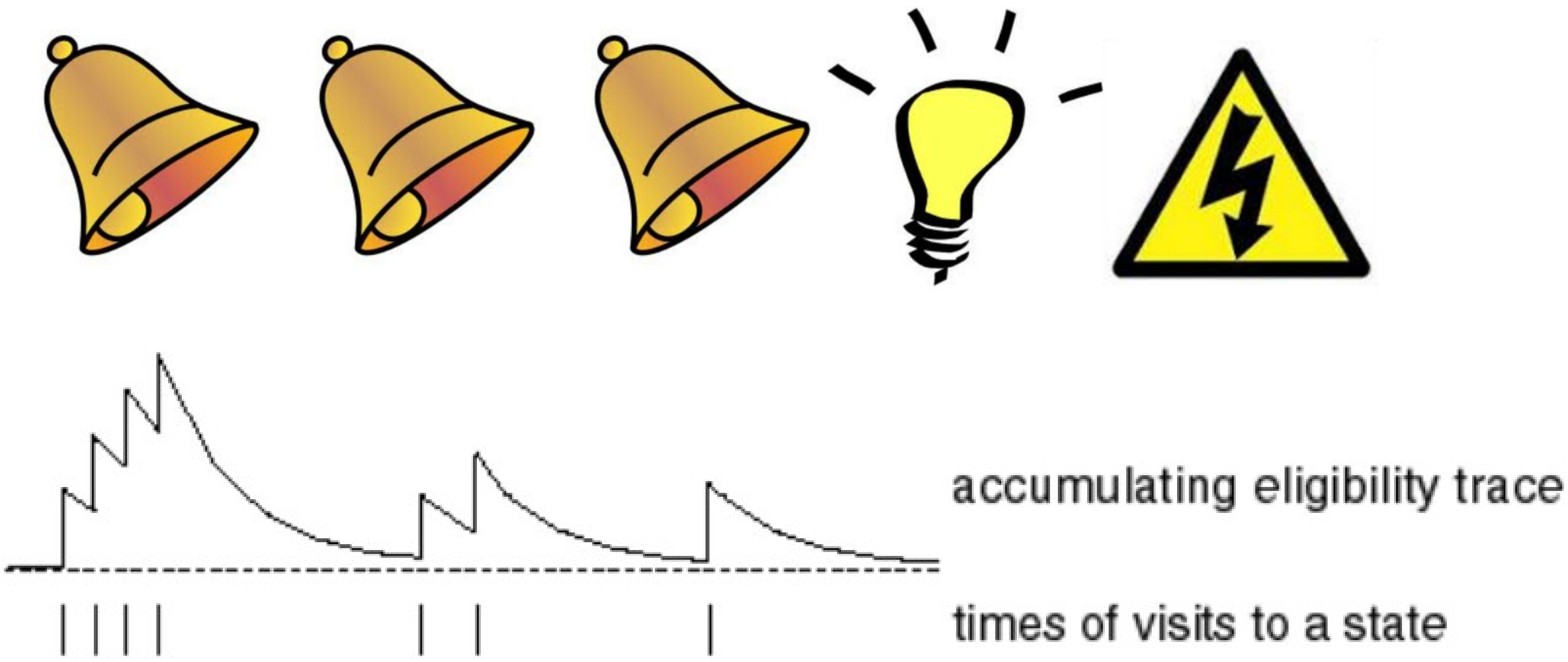
$$V = \Sigma_{t=0:\infty} \gamma^t R(s_t)$$

discounting has many benefits
(which is why it's almost always preferred in practice,
over finite horizon truncation)

it helps us deal with model uncertainty!

expected rewards far in the future assume that
the world will progress in the way you expect it to
for a noisy model (with compounding errors)
your prediction is unlikely to be accurate far into the future

it helps us deal with credit assignment problem!



it helps us avoid nonstartionary value functions!

it roughly approximates a finite horizon
(i.e. front weights rewards in time)
but will incorporate rewards for an infinite horizon
(i.e. no cutoff)

though to have an impact after discounting,
rewards that are infinitely far away,
must also be infinitely large!

what's the problem… ?

we still have to be able to look infinitely far ahead
to sum up all the possible rewards!

luckily we will never actually have to do this in practice!

since we iteratively update our value function,
we really only ever look one step ahead at a time

$$V = \Sigma_{t=0:\infty} \gamma^t R(s_t)$$

let's take this definition of value, and turn it into an iterative definition, called the Bellman Equation:

super useful!

$$\boxed{V_{k+1}(s_t) = R(s_t) + \gamma \max_{a_t} \Sigma_{s_{t+1}} P(s_{t+1} \mid s_t, a_t)\, V_k(s_{t+1})}$$

the expected value of a state after updating

is the observed reward gained in that state

plus your current expected reward for the best action you could take from that state, weighted by the probability of each possible new state resulting from that action

$$V(s_t) \leftarrow R(s_t) + \gamma \max_{a^t} \Sigma_{s^{t+1}} P(s_{t+1} \mid s_t, a_t)\, V(s_{t+1})$$

the Bellman Equation is an example of temporal difference (TD) learning

$$V(s) \leftarrow V(s) + \alpha\, (\, R(s) + \gamma\, V(s') - V(s)\,)$$

The variant of TD Learning that is used most often in practice is Q-learning

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

model-free way to look forward at successor states!

we'll revisit this in detail when we talk about deep reinforcement learning

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

One downside of Q-leaning is that you have to
take the max over all possible actions
(similar to what you've been doing with your game trees)

how could you remove the max operator?

you could sample from your
(stochastic or greedy) policy
and just update your Q function
with the information from that action

$$Q(s, a) = Q(s, a) + \alpha \, (\, R(s) + \gamma \, Q(s', \pi(s)') \; - Q(s, a) \,)$$

this algorithm is called SARSA
(for state → action → reward → state → action)

adds stochasticity to updates

you could change the algorithm to use
the mean expected value over all actions
instead of the best one

$$Q(s, a) = Q(s, a) + \alpha \left(R(s) + \gamma \, \mathrm{mean}_{a'} \, Q(s', a') - Q(s, a) \right)$$

this loses theoretical convergence guarantees
with policy iteration

but it does help to avoid bias overestimations of Q
due to noise