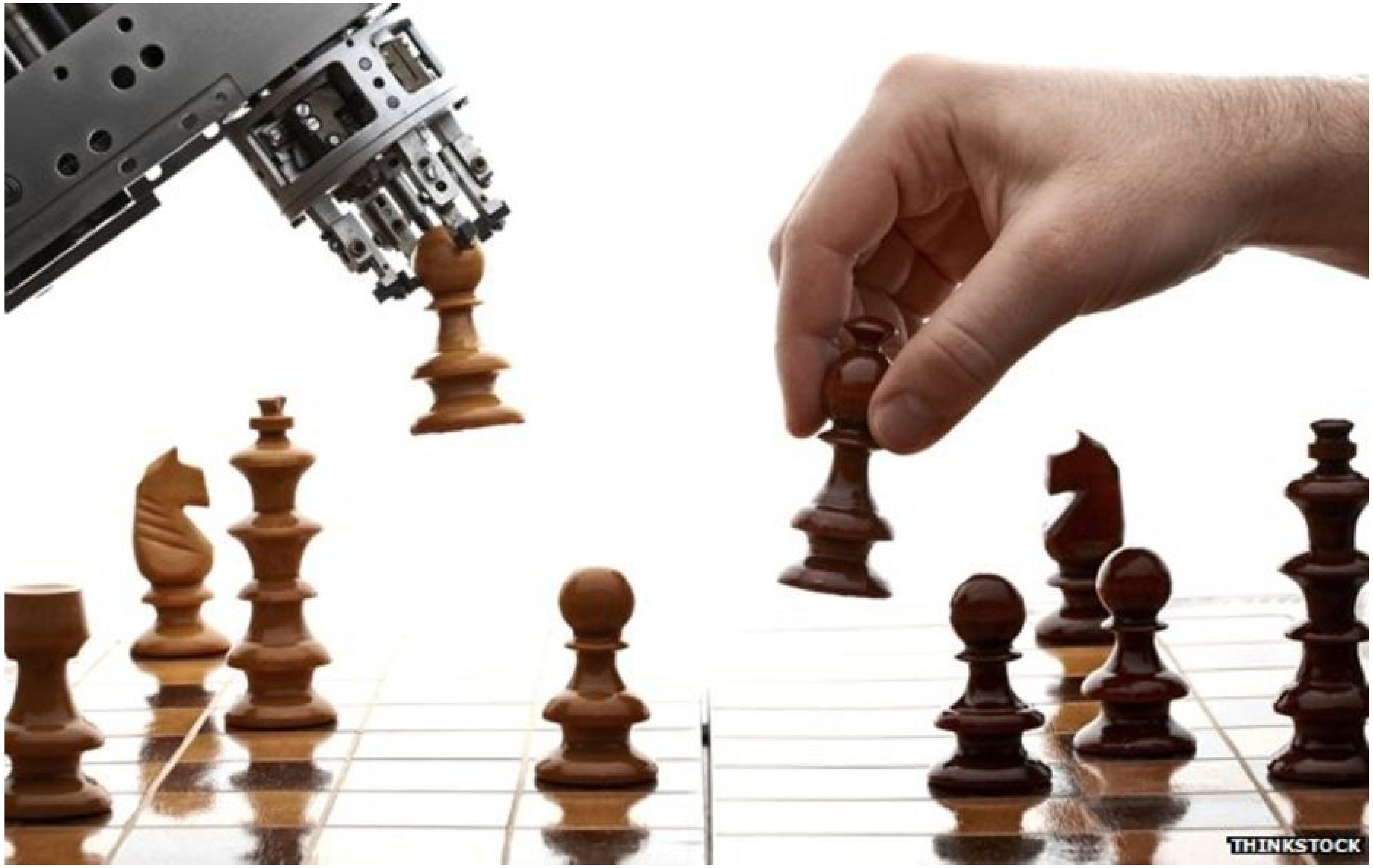


Introduction to Artificial Intelligence

COSC 4550 / COSC 5550

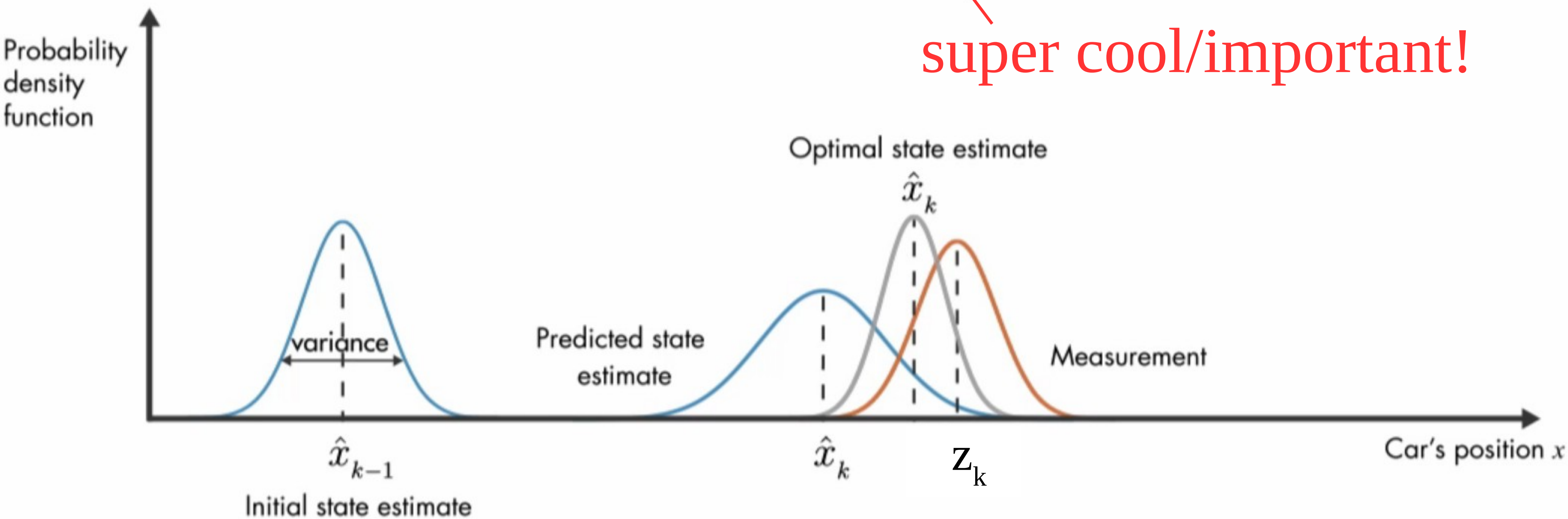
Professor Cheney

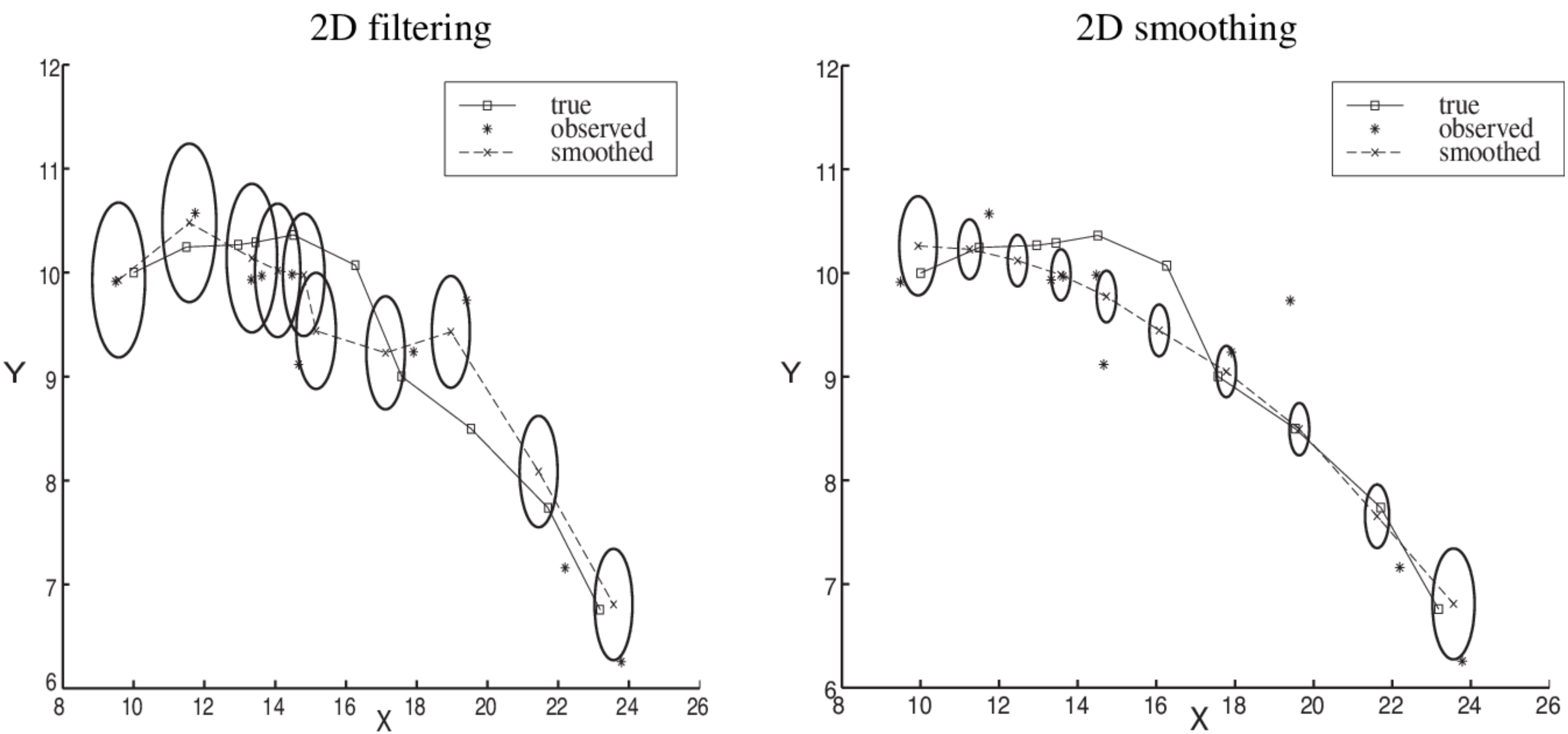
10/6/17

the variance of our new estimate is less than either our predicted state or our measurement (combining two estimates gives us more certainty!)

this is a property of multiplying Gaussian distribution (variance often grows when multiplying other distributions)

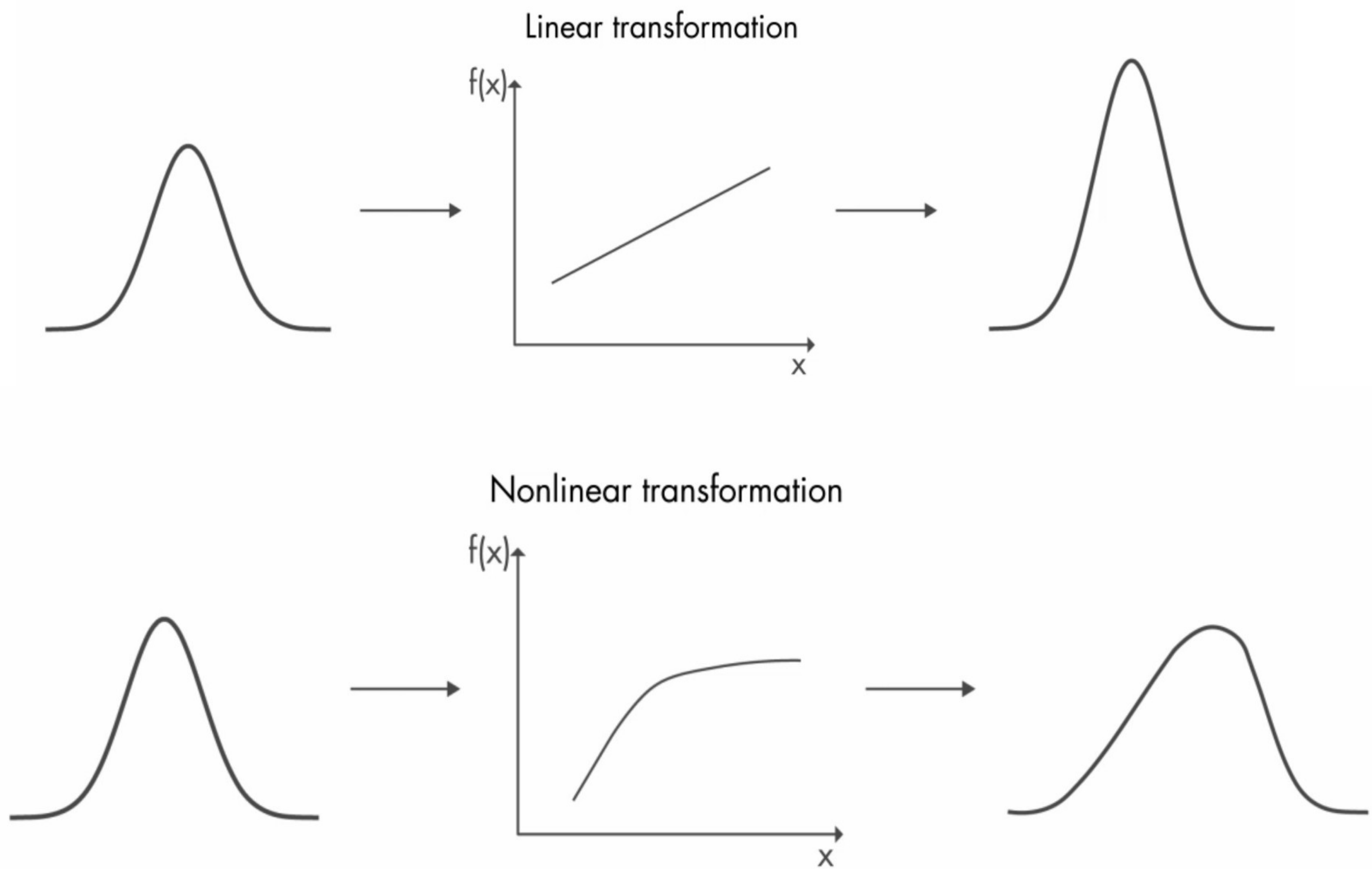
super cool/important!

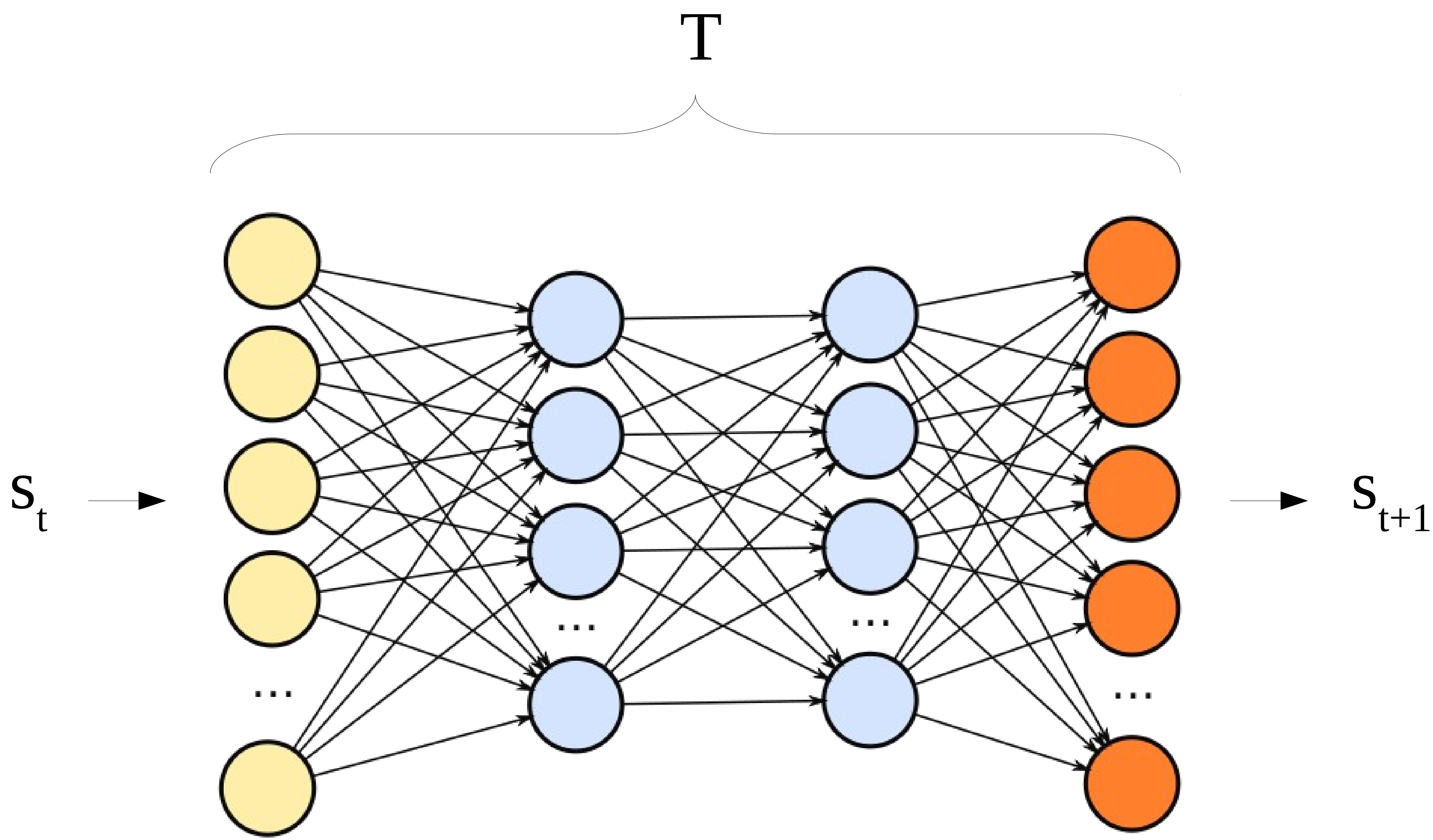


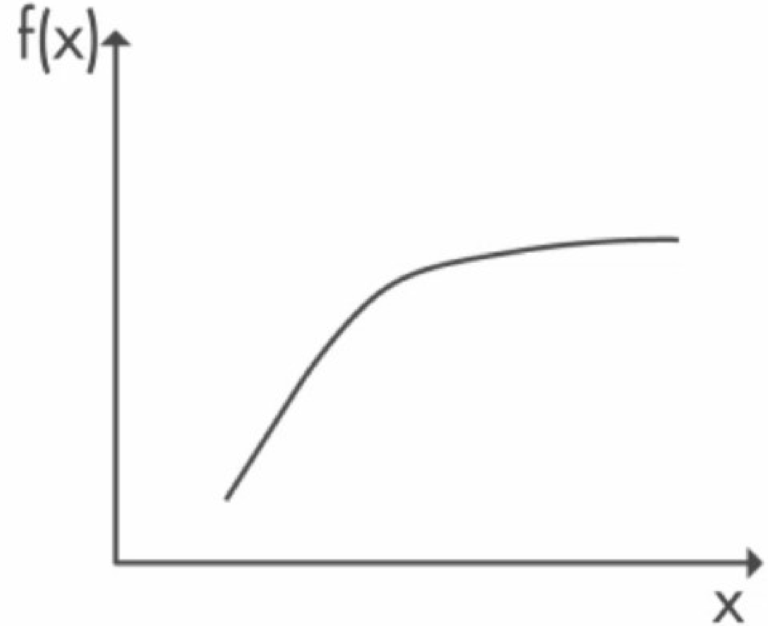

2D filtering
true
observed
smoothed
Y
X
2D smoothing
true
observed
smoothed
Y
X

one of the simplifying assumption we made
(that helped the Gaussian function to behave nicely)
is that our transition model was linear

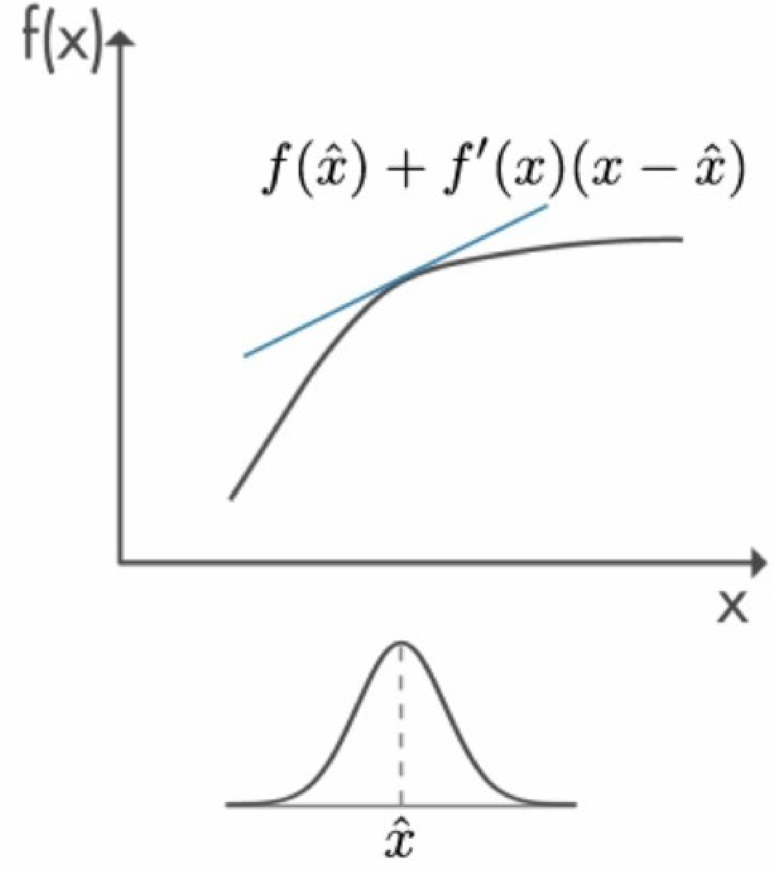
$$(X_{t+\Delta} = X_t + X' * \Delta)$$


Linear transformation
f(x)
x
Nonlinear transformation
f(x)
x


T
s_t
s_{t+1}
...
...
...
...

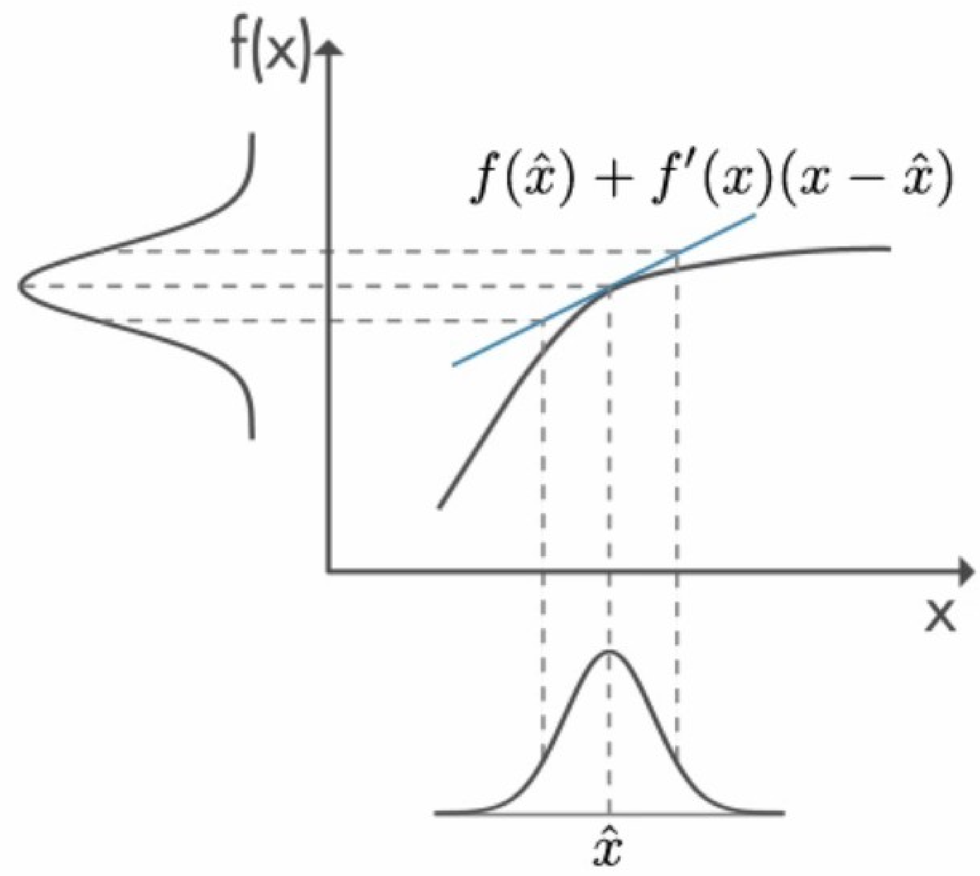


f(x)
x



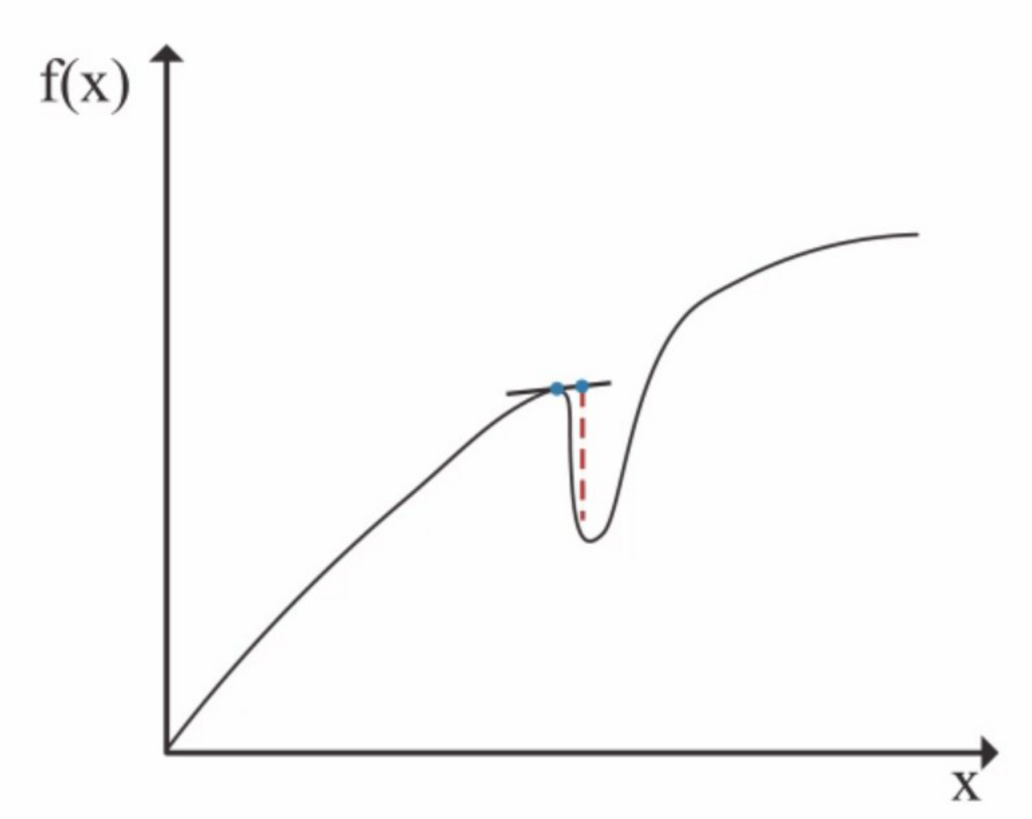
f(x)
$f(\hat{x}) + f'(x)(x - \hat{x})$
x
$\hat{x}$

“extended Kalman filter”

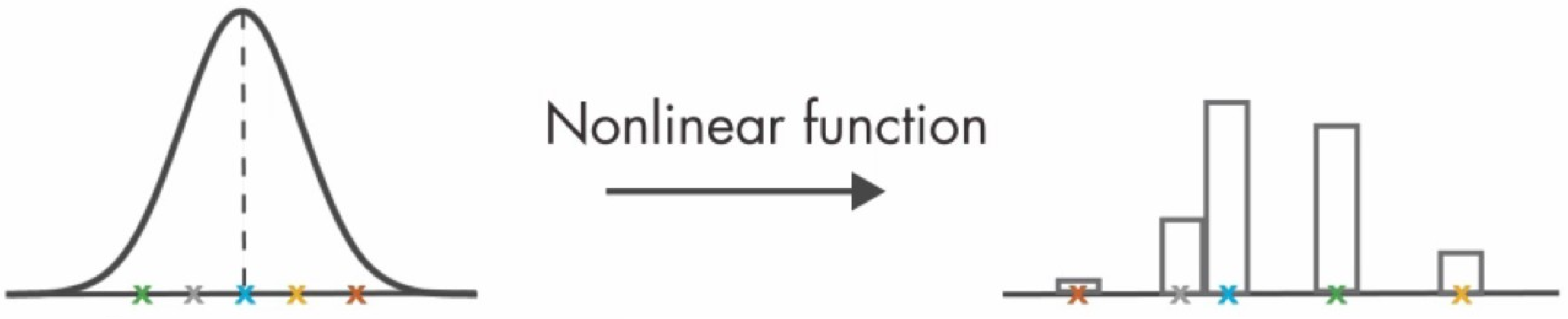


this is great whenever your
function is smooth/differentiable

but sometimes it's not...

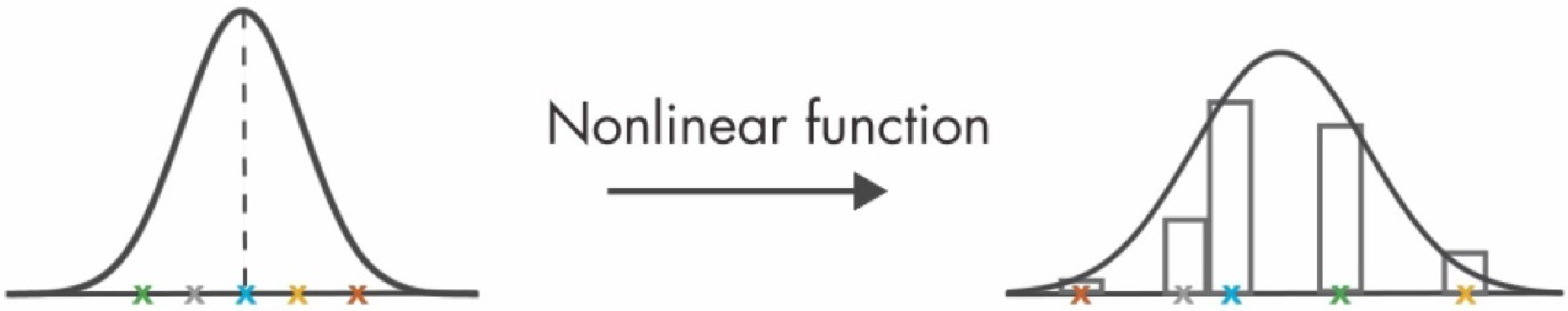


f(x)
x



let's sample points
from our distribution

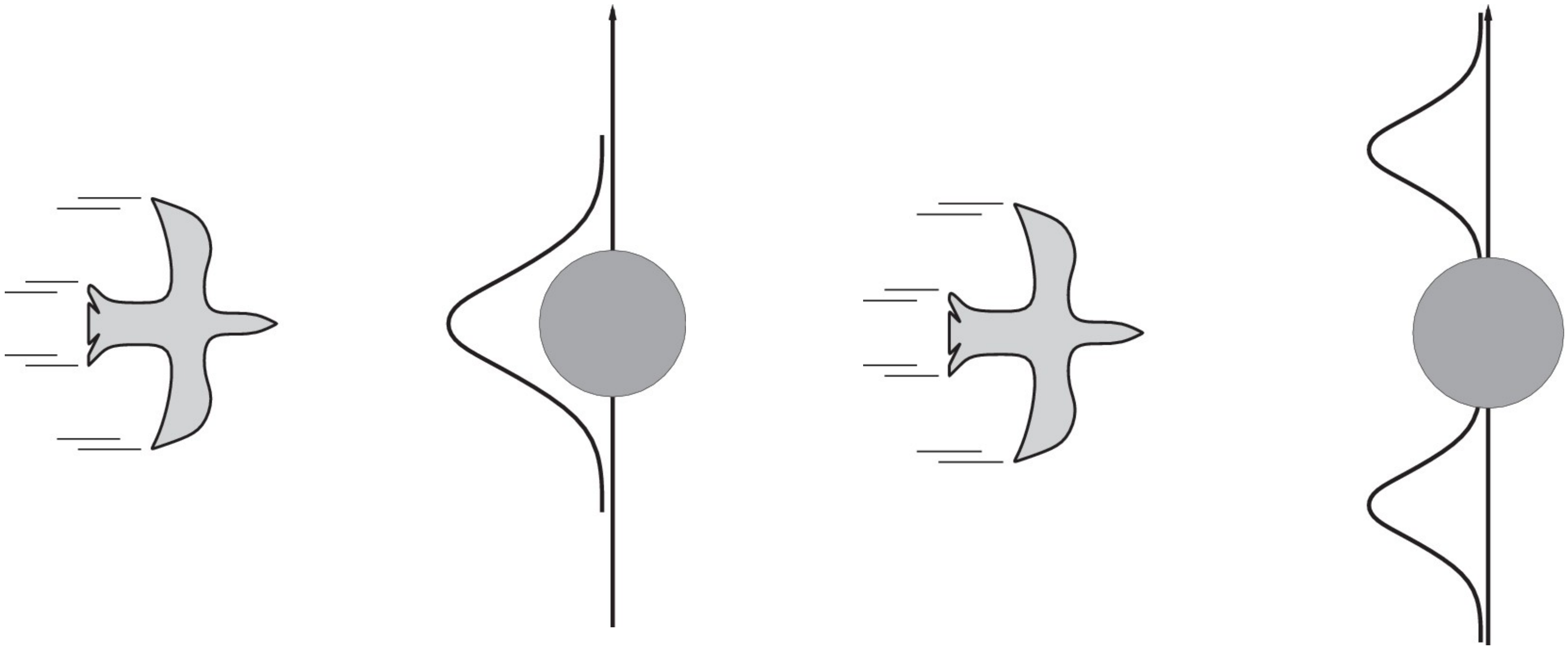
“particle filter”

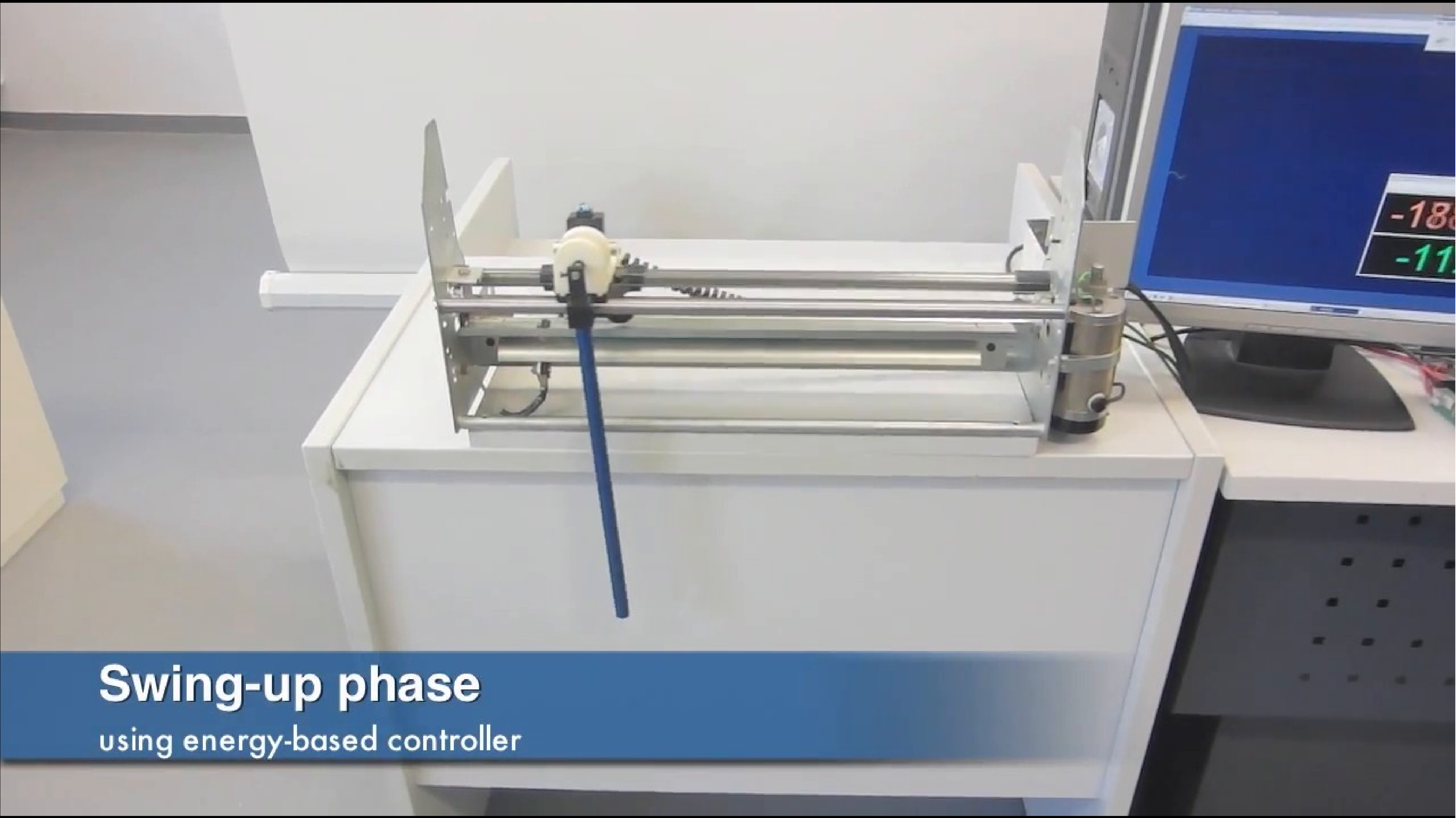


then fit a new Gaussian to those points after they go through our non-linear transformation

Kalman filters and HMMs are subsets of dynamic Bayesian networks (DBN)

but not all DBNs can be represented as a Kalman filter… e.g bifurcation events




Swing-up phase
using energy-based controller

reinforcement learning!

Part I: Artificial Intelligence

- Introduction
- Intelligent Agents

what are agents?
why would they need to do search?

Part II: Problem Solving

- Search
- Optimization
- Games

how do we even do search?
what's the best way to do it?

what if we have a simple setting, where we perfectly know the rules (i.e. model)?

~~**Part III: Knowledge, Reasoning, & Planning**~~

if we know things about the problem already, can we tell them to the agent, instead of making it learn them?

Part IV: Uncertainty and Reasoning

- Probability
- Bayesian Statistics
- Markov Models

if we're not sure of the model, but have a guess at how it should work, how can we update our causal understanding when new information comes?

Part V: Learning

- Unsupervised Learning
- Supervised Learning
- Reinforcement Learning

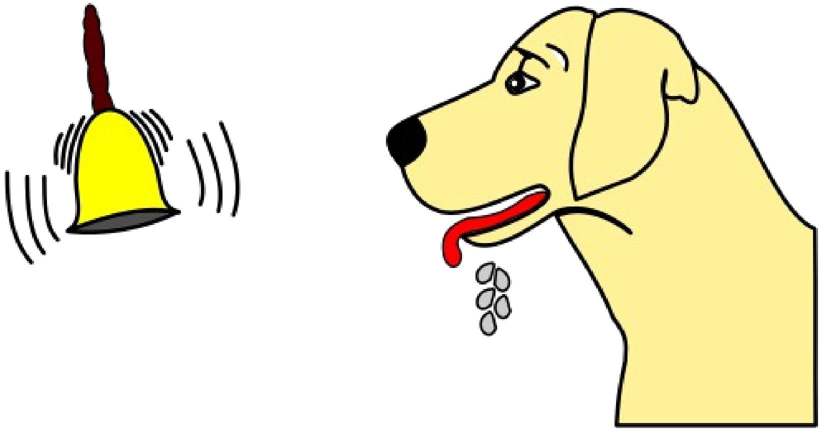
what if we have no idea (or prior assumptions) about how the world works – can we get the agent to learn correlations from the ground up?

Part VI: Communicating, Perceiving, & Acting

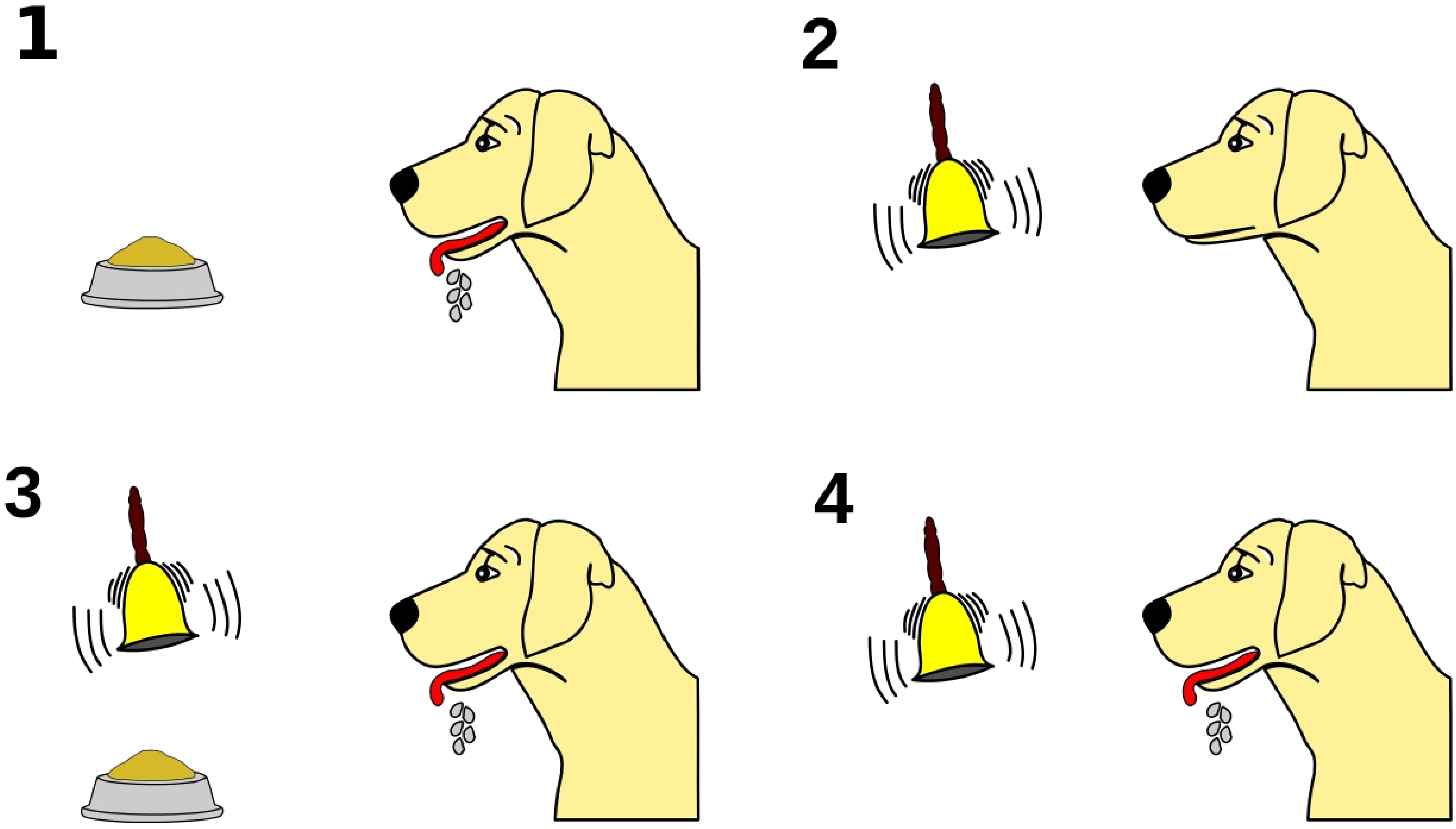
- Natural Language Processing
- Object Recognition
- Robotics

what additional tricks/techniques do we need to be able to apply these ideas to a variety of applications?

optimal decision making
in sequential decision making tasks

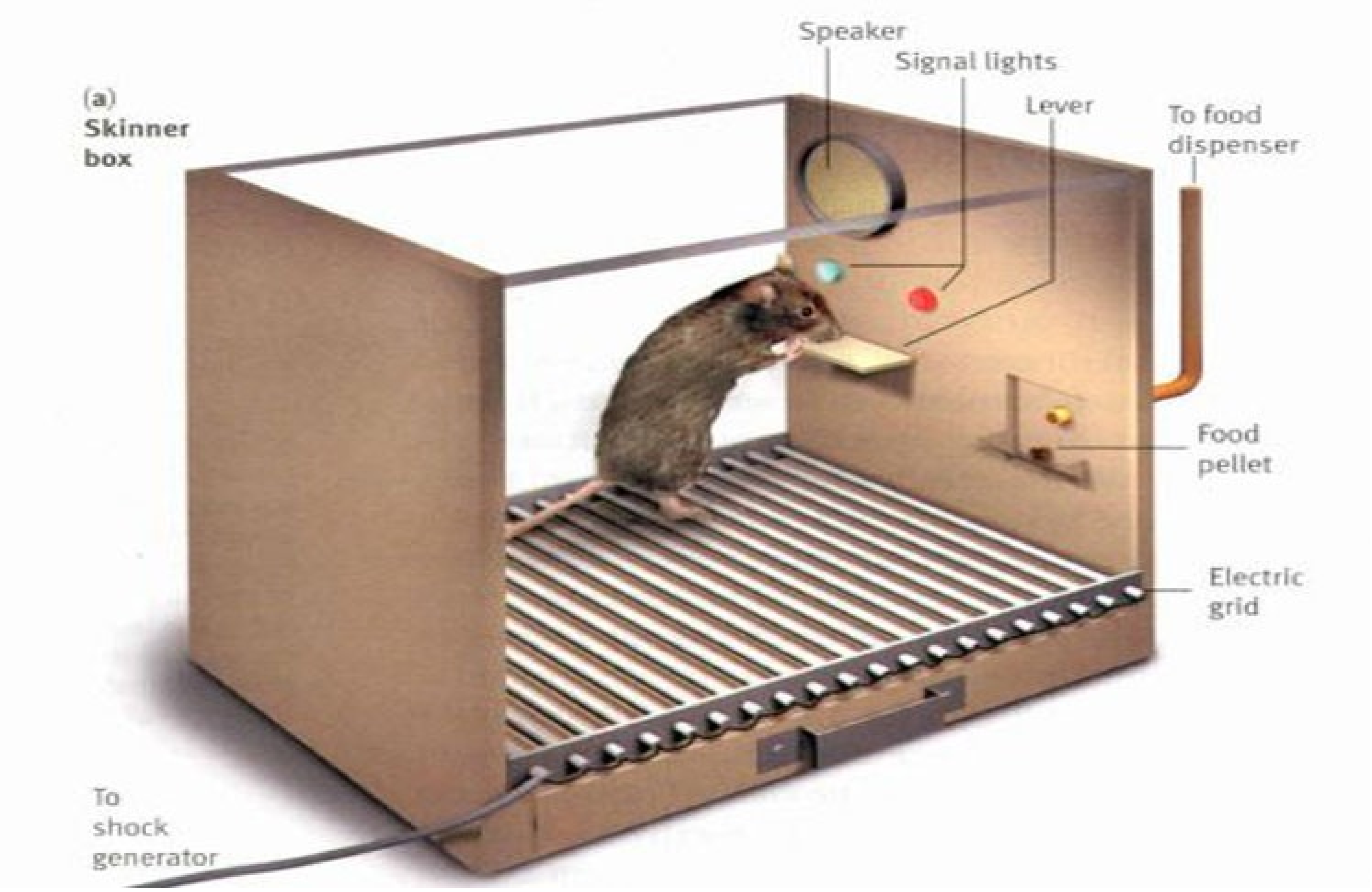


“classical conditioning”



“operant conditioning”

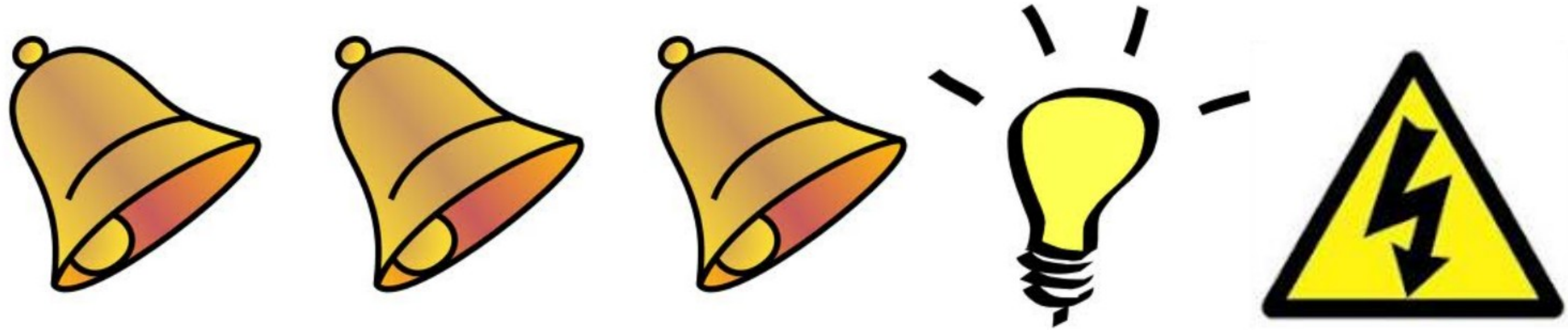
“operant conditioning”



basic idea:

optimize a behavioral **policy**,
such that the agent learns to
recognize **states** that is has been in
before, and reproduce the **actions**
that have led to positive **rewards**
in that state in past experiences

“credit assignment problem”



what caused the shock?

state (s)
action (a)
reward (R(s))

policy (π: s$\rightarrow$a)

policy is optimized to maximize cumulative reward
(V for “value” or U for “utility”)

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

$$state_i \rightarrow action_i \rightarrow reward_i \rightarrow state_{i+1}$$

value iteration

so far this semester, you've given an agent its value function

e.g.
straight-line distance heuristic in map search
manhattan distance heuristic in pacman
corners heuristic in pacman
actual A* # of moves left in pacman

we (humans) can design these (heuristic) value functions
because we intuitive know how the problem works
and what sort of strategies we might useful to solve it

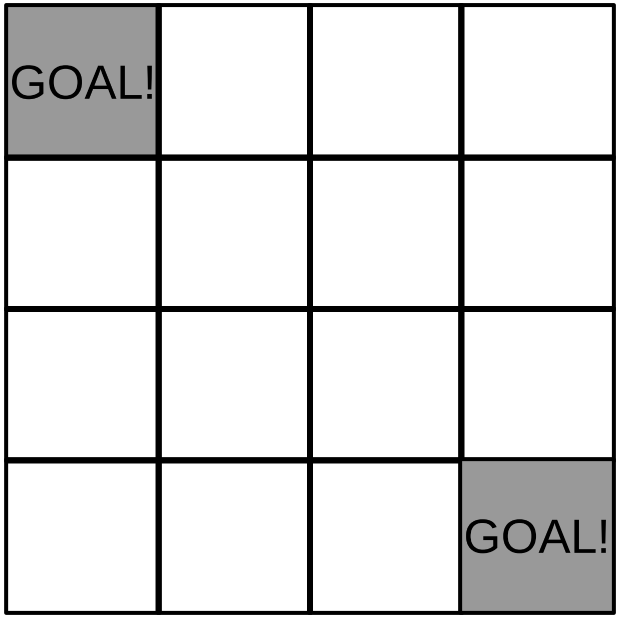
an artificial agent has no intuition,
it has to build up its knowledge of what
states are good or bad through experience
(i.e. iteratively) as it interacts with the world
(through some behavioral policy)

grid world:

each timestep has -1 reward

the game terminates when you reach a goal state

actions: N, S, E, W



intuitive description: “get to the goal as soon as possible”
(but let's pretend we're a robot, who doesn't know this!)

each value function (V) is defined
with respect to some behavioral policy (π)
V^{π}

let's iteratively find V^{π} for a random policy
in our mini grid world

current value (V_k) for a random policy

k=0
"who knows?"

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

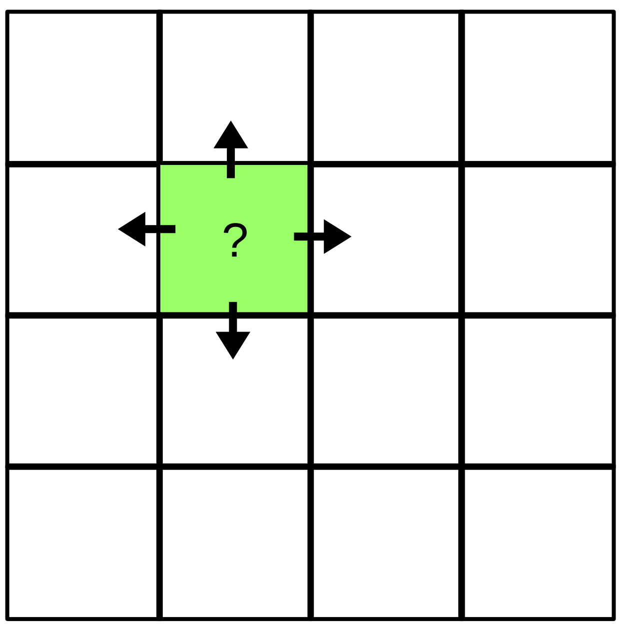
k=1

current value (V_k) for a random policy

k=0
“who knows?”

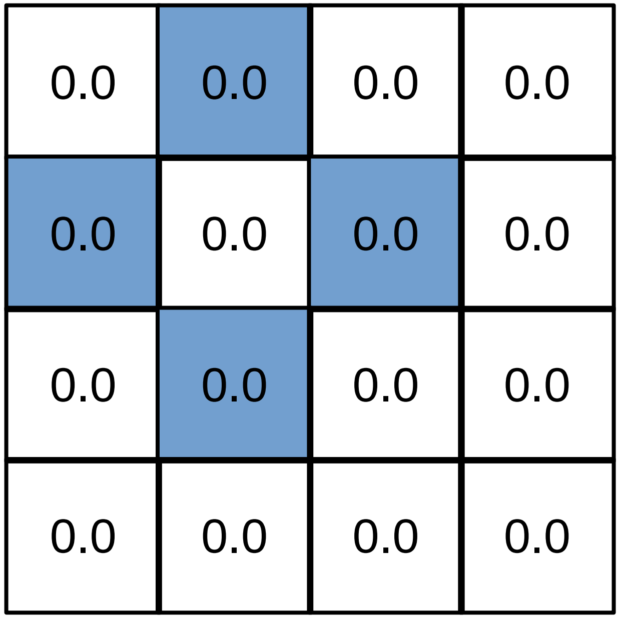
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

k=1

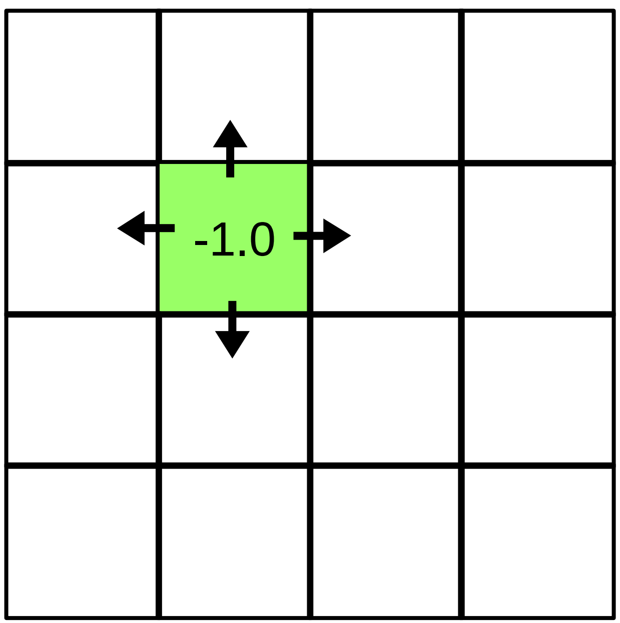


current value (V_k) for a random policy

k=0
"who knows?"



k=1



immediate reward

current prediction for cumulative reward in new state

N: $V_{s,N} = -1 + 0 = -1$

S: $V_{s,S} = -1 + 0 = -1$

E: $V_{s,E} = -1 + 0 = -1$

W: $V_{s,W} = -1 + 0 = -1$

with a random policy,
we are equally likely to take any move,
so:

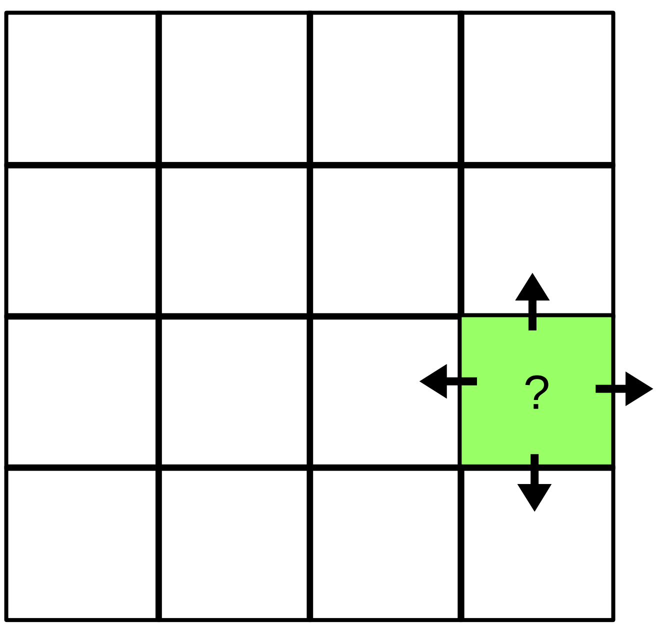
$V_s = (-1 + -1 + -1 + -1)/4 = -1$

current value (V_k) for a random policy

k=0
“who knows?”

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

k=1



current value (V_k) for a random policy

k=0
“who knows?”

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

k=1

			-1.0

current value (V_k) for a random policy

k=0
"who knows?"

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

k=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

next iteration…
new value function
becomes
old value function
("current prediction
for cumulative reward")

current value (V_k) for a random policy

k=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

k=2

current value (V_k) for a random policy

k=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

k=2

	-1.75		

N: $V_{s,N} = -1 + -1 = -2$

S: $V_{s,S} = -1 + -1 = -2$

E: $V_{s,E} = -1 + 0 = -1$

W: $V_{s,W} = -1 + -1 = -2$

$V_s = (-2 + -2 + -1 + -2)/4 = -1.75$

current value (V_k) for a random policy

k=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

k=2

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0

current value (V_k) for a random policy

k=2

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0

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

current value (V_k) for a random policy

k=10

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

k=∞

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

converged to true value function ($V^{\pi\text{-random}}$)

NATURE | VOL 529 | 28 JANUARY 2016

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

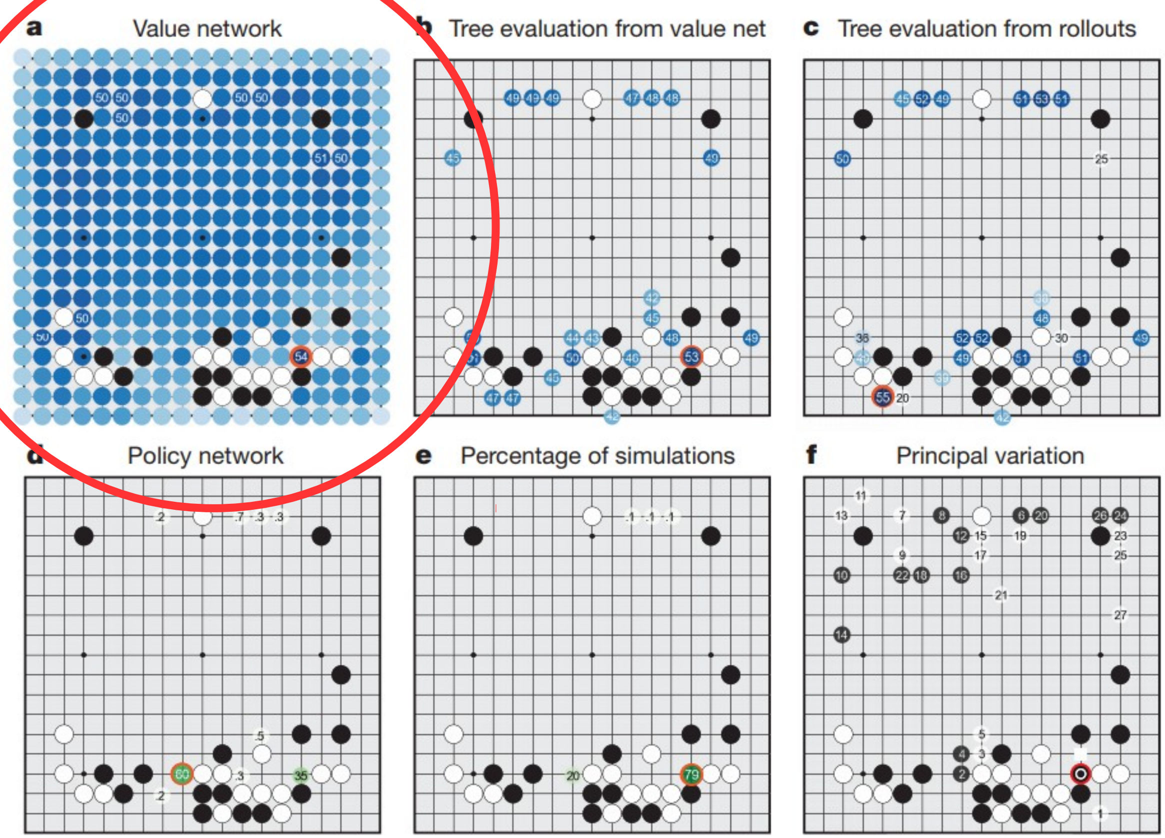


Figure 5 | How AlphaGo (black, to play) selected its move in an informal game against Fan Hui. For each of the following statistics, the location of the maximum value is indicated by an orange circle. **a**, Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b**, Action values $Q(s, a)$ for each edge (s, a) in the tree from root position s; averaged over value network evaluations only ($\lambda = 0$). **c**, Action values $Q(s, a)$, averaged over rollout evaluations only ($\lambda = 1$). **d**, Move probabilities directly from the SL policy network, $p_\sigma(a|s)$; reported as a percentage (if above 0.1%). **e**, Percentage frequency with which actions were selected from the root during simulations. **f**, The principal variation (path with maximum visit count) from AlphaGo's search tree. The moves are presented in a numbered sequence. AlphaGo selected the move indicated by the red circle; Fan Hui responded with the move indicated by the white square; in his post-game commentary he preferred the move (labelled 1) predicted by AlphaGo.

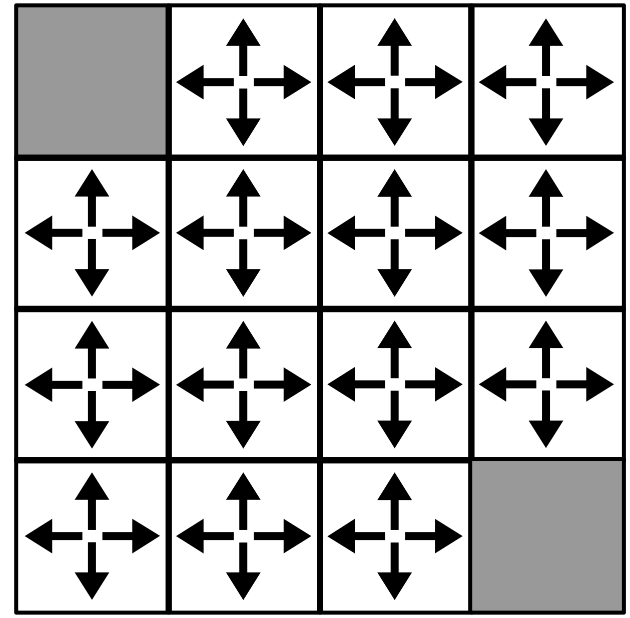
let's use our value function to produce a (greedily) optimal policy!

current value (V_k) for a random policy

greedy policy (π_k) for a this value function

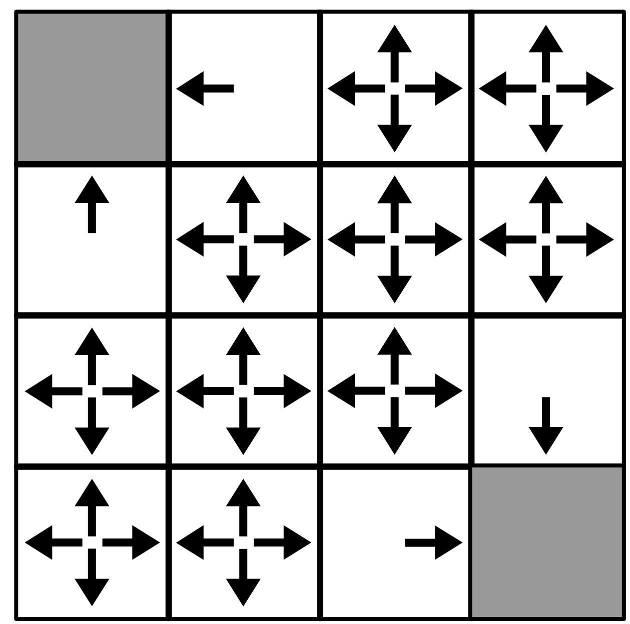
k=0
“who knows?”

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



k=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

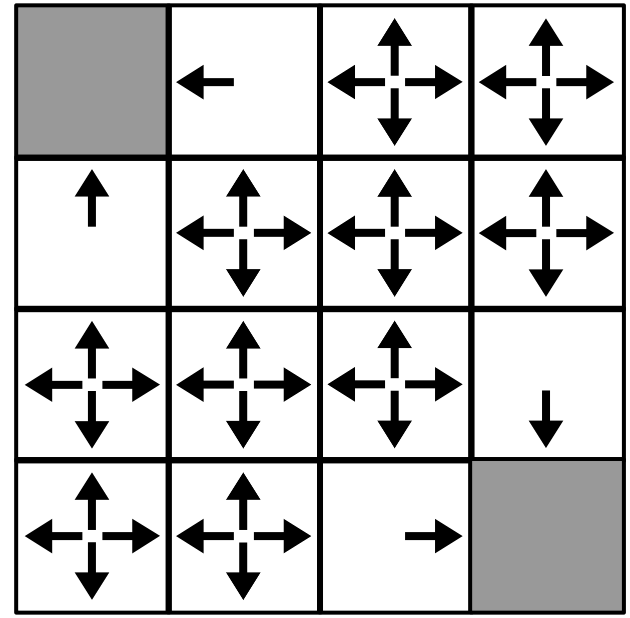


current value (V_k) for a random policy

greedy policy (π_k) for a this value function

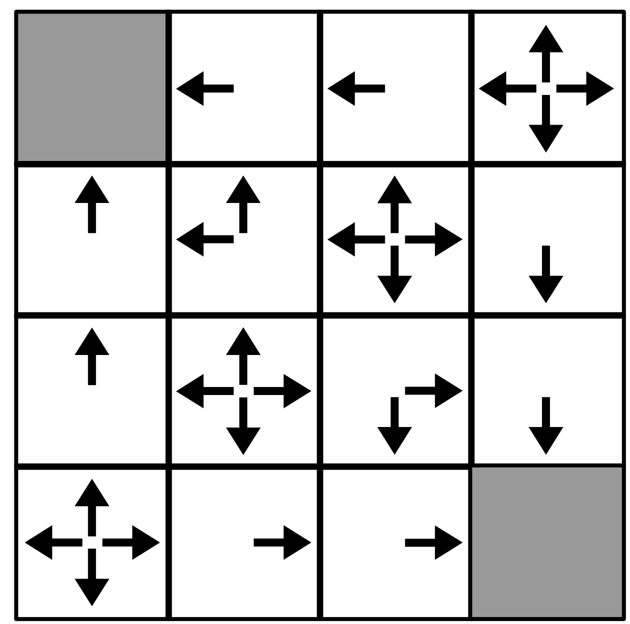
k=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



k=2

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0

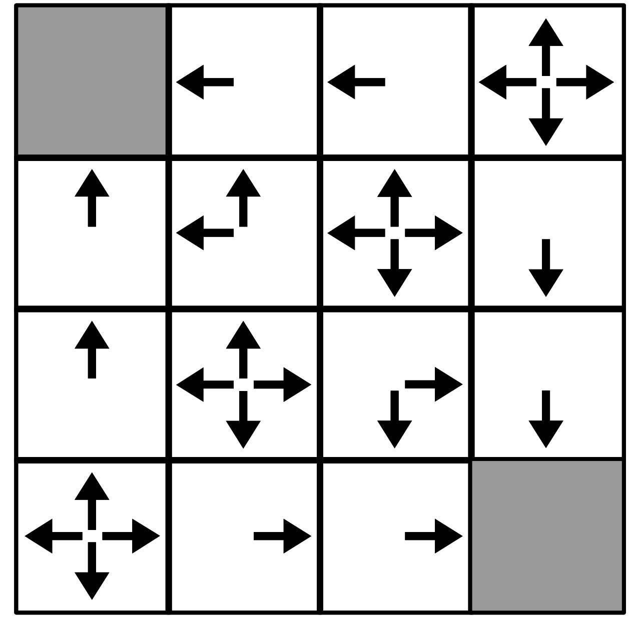


current value (V_k) for a random policy

greedy policy (π_k) for a this value function

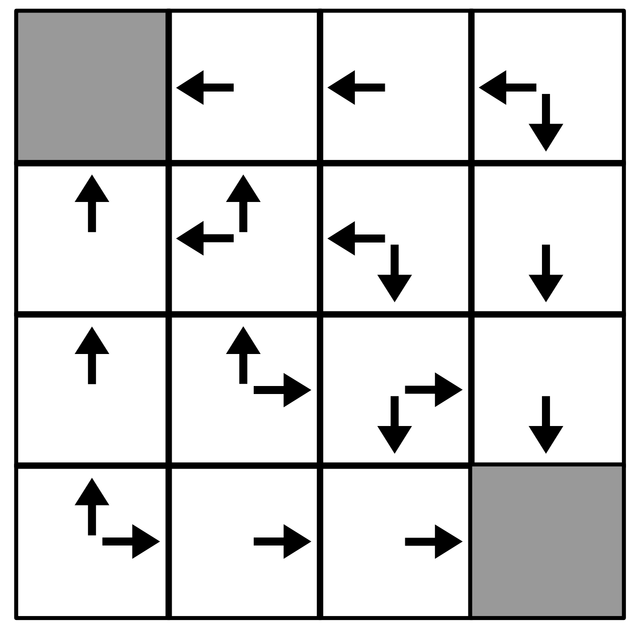
k=2

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0



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

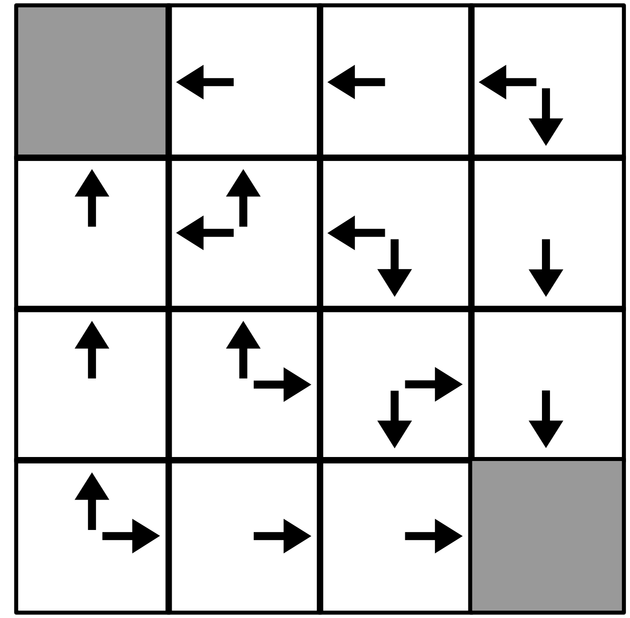


current value (V_k) for a random policy

greedy policy (π_k) for a this value function

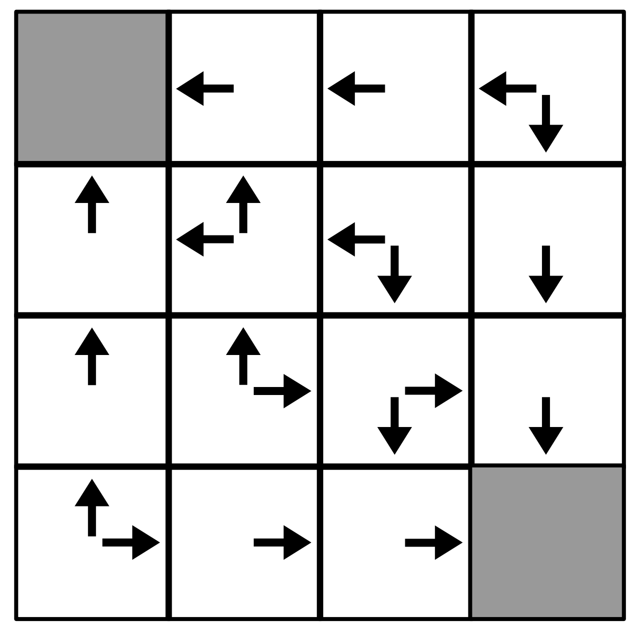
$k=10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



$k=\infty$

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



the greedy policy converges faster than the value function!

(all we need to find the best policy for a value function is the correct ordering of states and not their exact values)

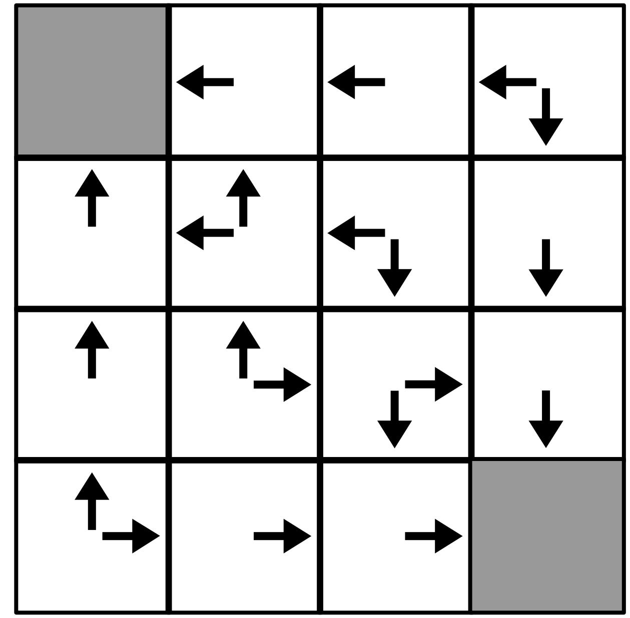
in this case, the policy was converged by k=3

current value (V_k) for a random policy

greedy policy (π_k) for a this value function

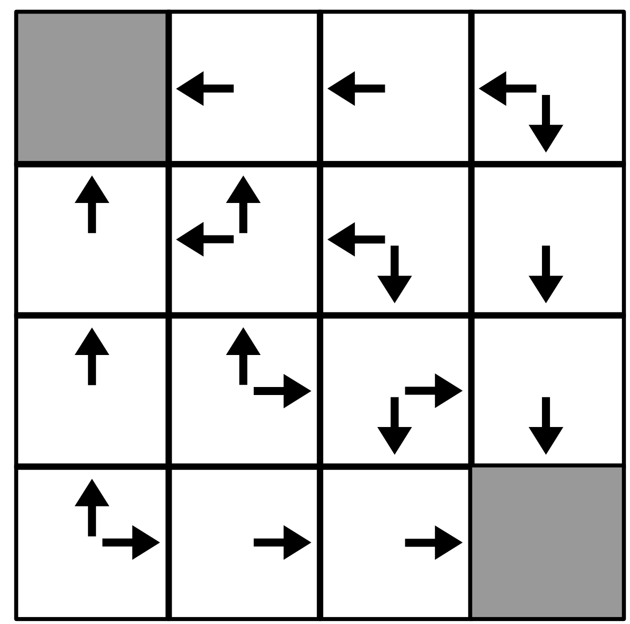
k=3

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

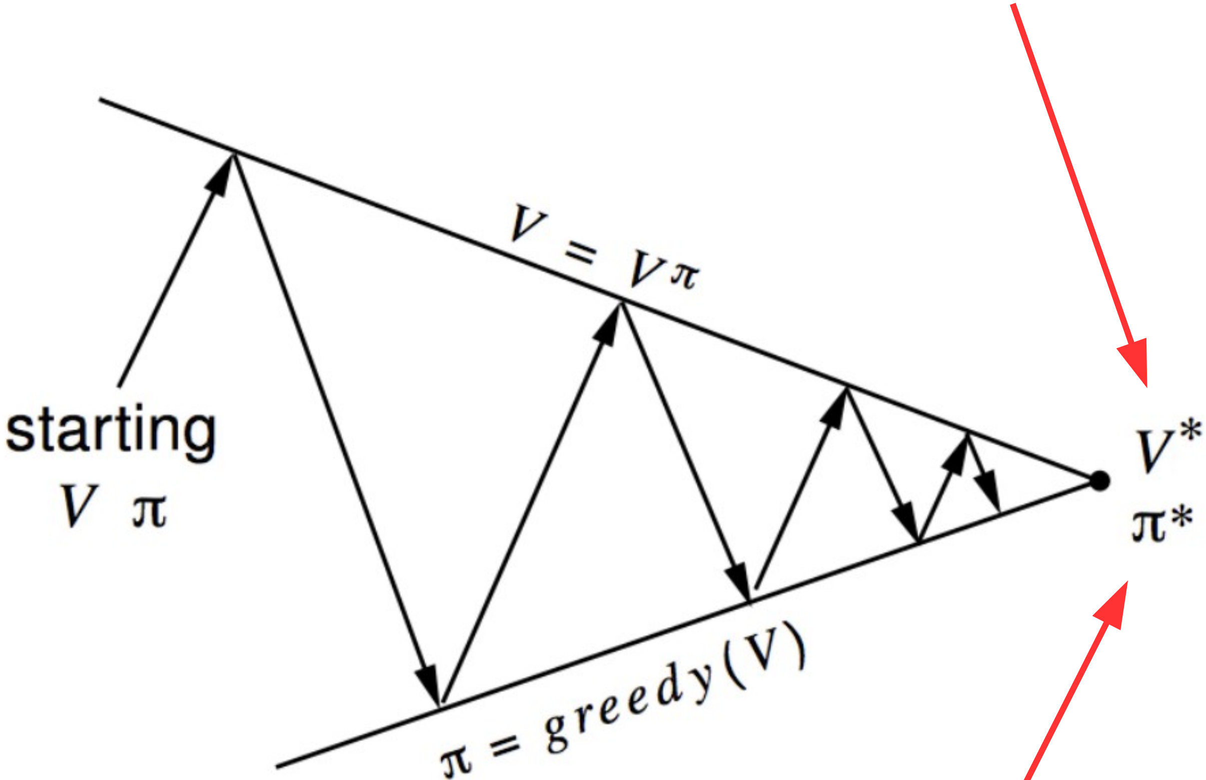


what could we do to improve this even further?

now that we have a better (non-random) policy, let's use our current (greedy) policy to choose our action weighting iterate through the value function

“on-policy learning”

optimal value function!



optimal policy!