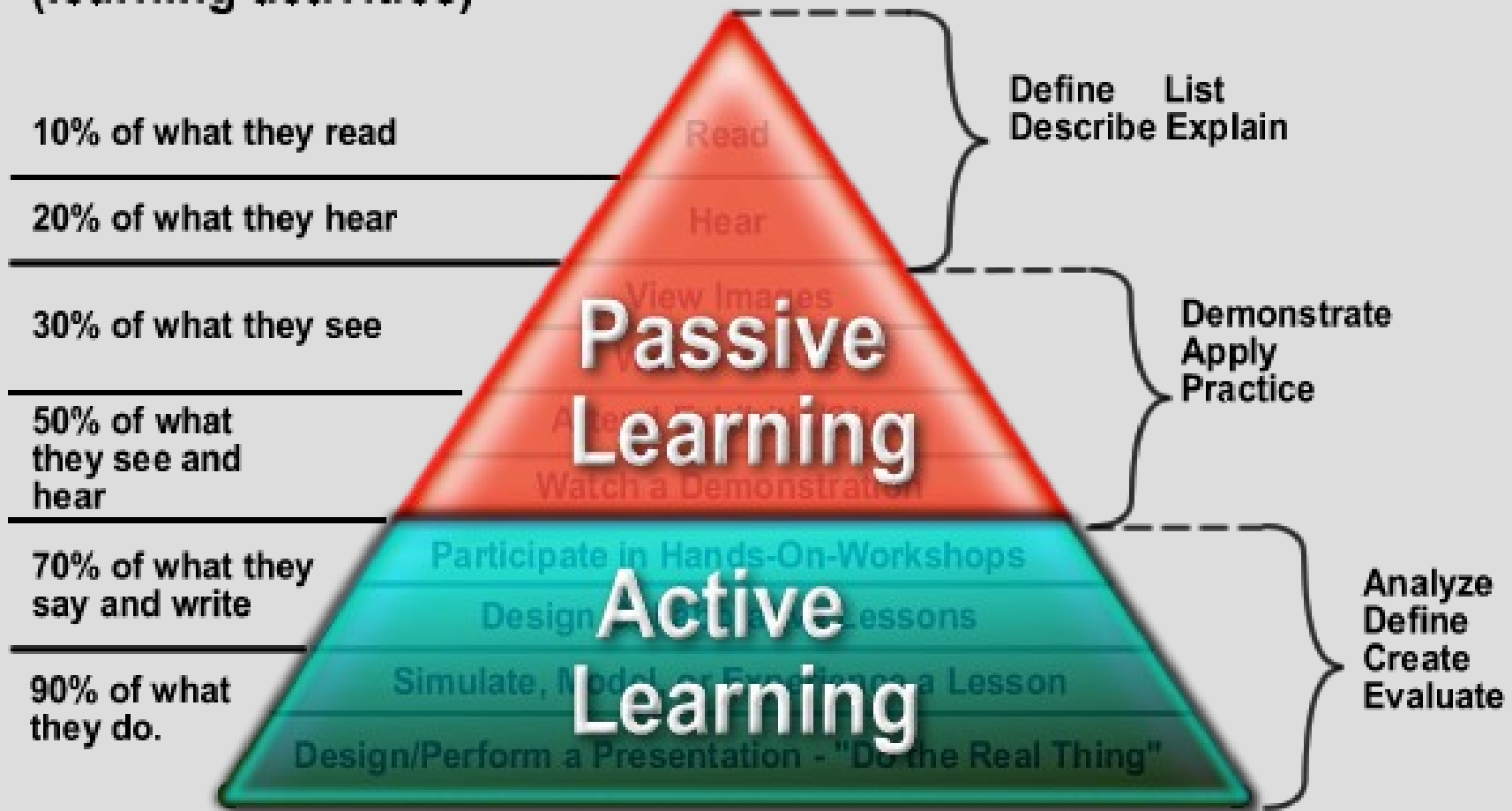


# Passive Learning (Ch. 21.1-21.2)

People generally remember...  
(learning activities)

People are able to...  
(learning outcomes)



# Reinforcement Learning

So far we have had labeled outputs for our data (i.e. we knew the homework was easy)

We will move from this (supervised learning) to where we don't know the correct answer, just if it was good/bad (reinforcement)

This is much more useful in practice as for hard problems we often don't know the correct answer (else why'd we ask the computer?)

# Reinforcement Learning

We will start by looking at passive learning, where we will not be taking actions, but just observing outcomes (because easier)

Next time we will move into active learning, where we can choose how we want to act to find the best outcomes/learn quickly

For now we want something we can observe, but see outcomes (i.e. rewards) for actions

# Reinforcement Learning

To do this, we will go back to our friend MDP

	T		
	↑	←	←
T	→		↑
	↑	←	↑

However since this is passive learning,  
we will only use the actions/arrows shown

(T's are terminal states, so no actions)

# Reinforcement Learning

How is this different than before?

(1) Rewards of states not known

(2) Transition function not known  
(i.e. no 80%, 10%, 10%)

Instead we will see examples  
of the MDP being run  
and learn the utilities

	T		
	↑	←	←
T	→		↑
	↑	←	↑

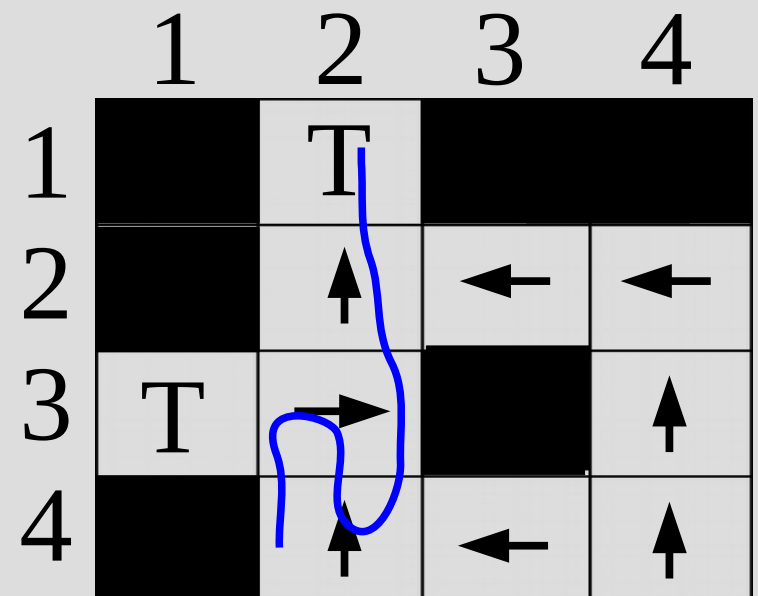
# Reinforcement Learning

Suppose we start in bottom row, left-most column and take the path shown

This will be recorded as (state)<sub>reward</sub>:

$(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (4,2)_{-1}$   
 $\uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}$

... then repeat this for more examples to better learn



# Direct Utility Estimation

$(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}$

The first (of three) ways to do passive learning is called direct utility estimation using reward:

$$U(s_0, s_1, s_2, \dots) = E \left[ \sum_{i=0}^{\infty} \gamma^i \cdot R(s_i) \right] \leftarrow \text{assume } \gamma=1 \text{ for simplicity}$$

Given this sequence, we can calculate the rewards at each step (starting from end):

(1,2) has reward 50

Then (2,2) is one less, so  $50-1 = 49$ ... so on

# Direct Utility Estimation

This gives us:

$(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}$   
45            46            47            48            49            50

Then we just find the average reward  
(4,2) visited twice (45,47)... average = 46

... and so on

(1,2) visited once... average reward = 50

Then update averages with future examples



# Direct Utility Estimation

So let's say you go straight to goal:

$$(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}$$

47            48            49            50

Then we update old averages with new data (only need store counts):

(4,2) visited once (47)... new average = 47

$$avg_{total} = \frac{count_{old} \cdot avg_{old} + count_{new} \cdot avg_{new}}{count_{old} + count_{new}} = \frac{2 \cdot 46 + 47}{3} = 46.33$$

(1,2) visited once... new average = 50,

so running total average now  $(50+50)/2=50$

# Direct Utility Estimation

Given that we are sampling the actions, this should lead to the correct expected utilities just by simple average


(This is not quite full reinforcement learning, as the actions are fixed so just learning utility)

But we can speed this up (i.e. learn much faster) by using some information

What info have we not used?

# Adaptive Dynamic Prog.

We didn't include our bud Bellman!

$$U(s) = \underbrace{R(s)}_{\text{rewards}} + \gamma \cdot \sum_{s'} \underbrace{P(s'|a, s)}_{\text{transition}} \cdot U(s')$$


no max over actions (a),  
as in passive actions are fixed

Thus, if we can learn the rewards and transitions, we can use our normal ways of solving MDPs (value/policy iteration)

This is useful as we can combine information across different states for faster learning

# Adaptive Dynamic Prog.

So given the same first example:

$$(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}$$

We'd estimate the following transitions:

$$(4,2) + \uparrow = 100\% \uparrow (2 \text{ of } 2)$$

$$(3,2) + \rightarrow = 50\% \uparrow, 50\% \downarrow$$

$$(2,2) + \uparrow = 100\% \uparrow$$

... and we can easily see the rewards from sequence, so policy/value iteration time!

 better as actions fixed no iteration

# Adaptive Dynamic Prog.

This method is called adaptive dynamic programming

Using the relationship between utilities (i.e. neighbors cannot change too much) allows us to learn quicker

This can be sped up even more if we assume all actions have the same outcome (i.e. going “up” has same probability for any state)

# Temporal-Difference

The third (last) way of doing passive learning is temporal-difference learning

↖ temporal = “time”

This is a combination of the first two methods, we will keep a “running average” of each state’s utility, but also use Bellman equation

Instead of directly averaging rewards to find utility, we will incrementally adjust them using the Bellman equation

# Temporal-Difference

Suppose we saw this example (bit **different**):  
 $(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}$

Using the direct averaging we would get:  
 $U(4,2) = 45, U(3,2) = 47$

However the sample(s) so far:  $(4,2) \uparrow$  is  
always  $(3,2)$ , so we'd expect (from Bellman):

$$U(4,2) = -1 + U(3,2)$$

# Temporal-Difference

This would indicate our guess of  $U(4,2)=45$  is a bit low (or  $U(3,2)$  is a bit high)

So instead of direct average, we will do incremental adjustments using Bellman:

$$U(s) \leftarrow U(s) + \alpha \cdot (R(s) + \gamma \cdot U(s') - U(s))$$

learning rate/constant

So whenever you take an action, you update the utility of the state before the action (final terminal state does not need updating)



# Temporal-Difference

Let's continue our example:

$$(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (3,2)_{-1} \rightarrow (2,2)_{-1} \uparrow (1,2)_{50}$$

So from first example:  $U(4,2)=45$ ,  $U(3,2)=47$

If second example starts as:

$$(4,2)_{-1} \uparrow (3,2)_{-1} \rightarrow \dots$$

could use TD learning on first example too... new states have  $U(s) = R(s)$ , then do updates as described

We'd update  $(4,2)$  as: (assume  $\alpha=0.5$ )

$$U(4,2) \leftarrow U(4,2) + \alpha \cdot (R(4,2) + \gamma \cdot U(3,2) - U(4,2))$$

$$\leftarrow 45 + \alpha(-1 + 1 \cdot 47 - 45)$$

$$\leftarrow 45.5$$

# Recap: Passive Learning

What are pros/cons between the last two methods? (adapt. dyn. prog. vs temporal-diff.)

Which do you think is faster at learning in general?

# Recap: Passive Learning

What are pros/cons between the last two methods? (adapt. dyn. prog. vs temporal-diff.)

-Temporal-difference only changes a single value for each action seen

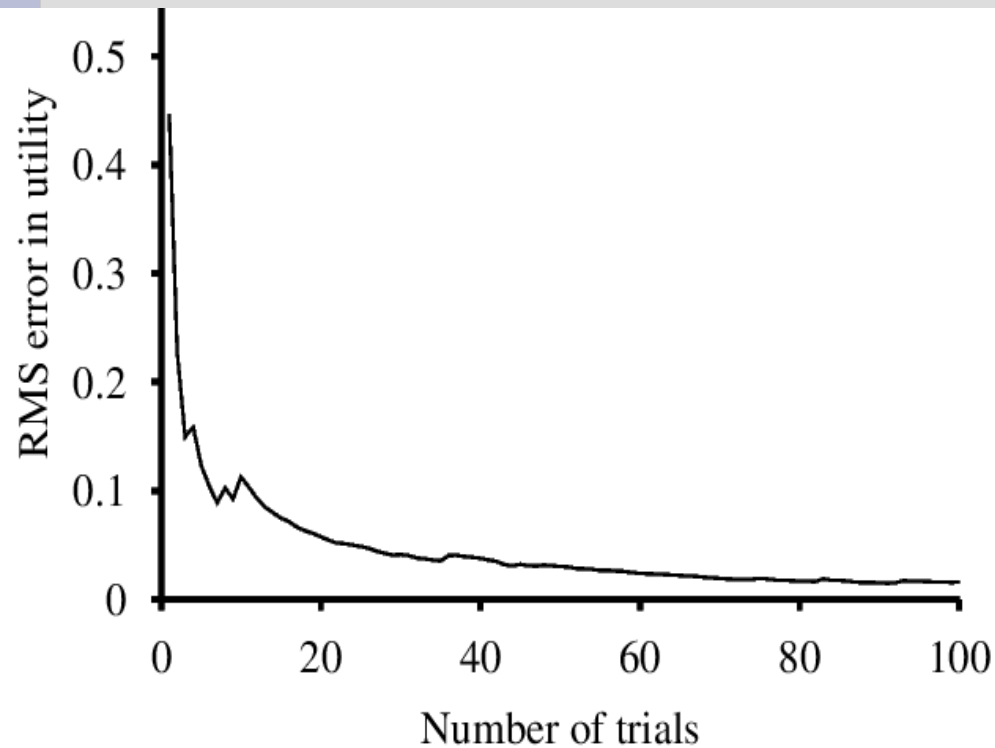
-ADP would re-solve a system of linear equations (policy “iteration”) for each action

Which do you think is faster at learning in general?

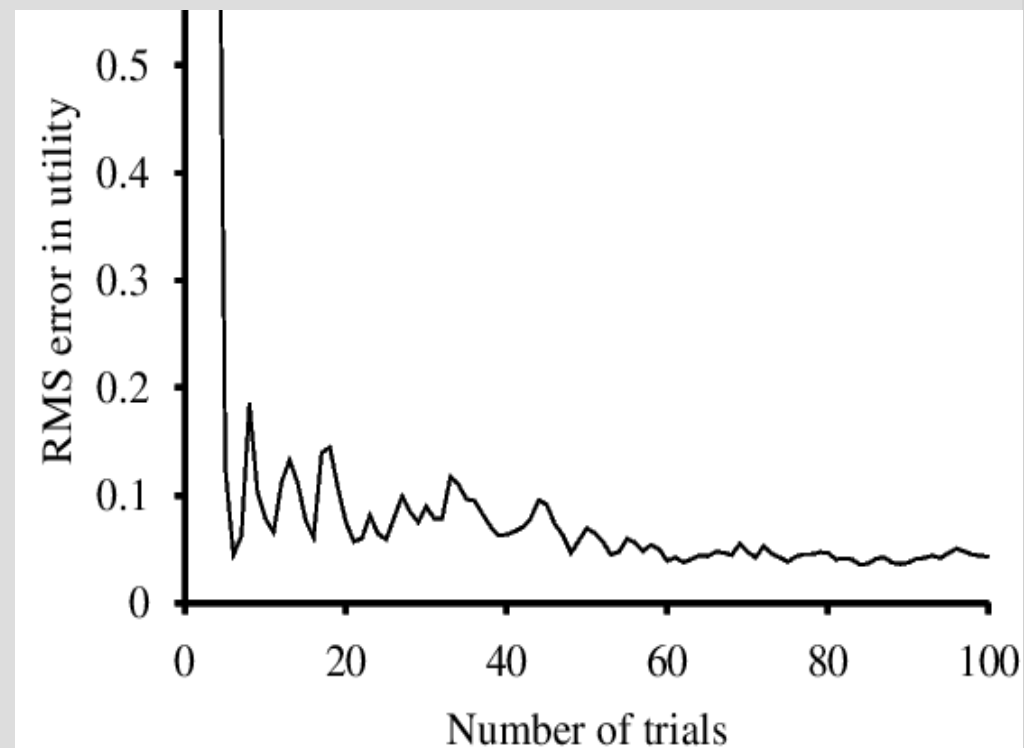
As ADP uses Bellman equations/constraints in full it learns better (but more computation)

# Recap: Passive Learning

From the book's example:



ADP



TD