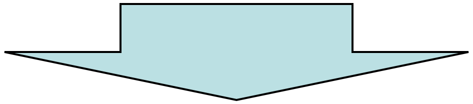
Sequential Decisions With Uncertainty: When Agents Learn

With slides from Pieter Abbeel and Dan Klein (Berkeley), and Percy Liang (Stanford)

HMMs for NLP: Tagging

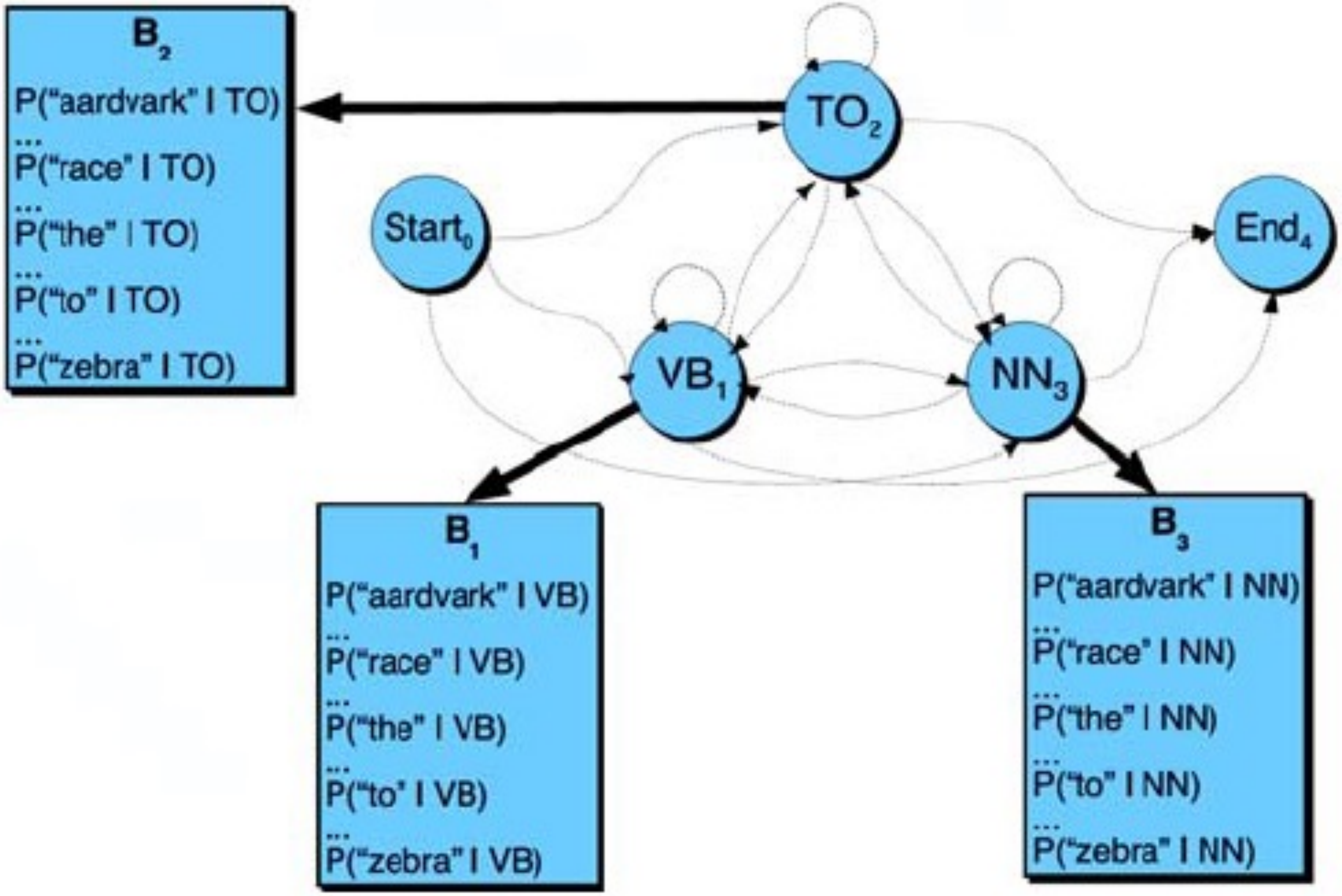
The Georgia branch had taken on loan commitments …



DT NNP NN VBD VBN RP NN NNS

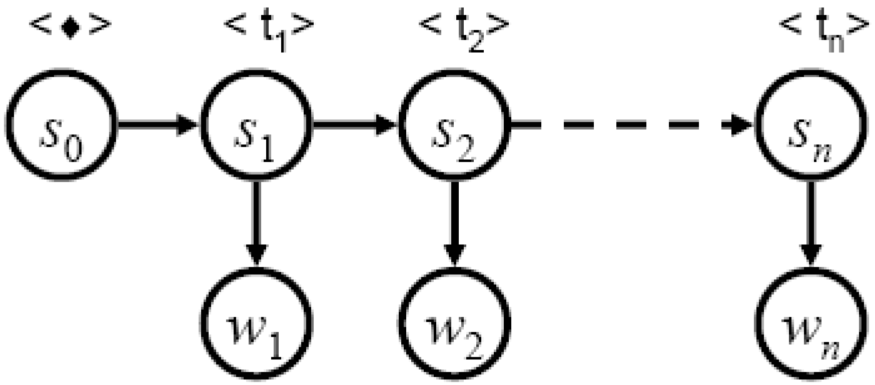
- HMM Model:
  - States Y = {DT, NNP, NN, ... } are the POS tags
  - Observations X = V are words
  - Transition dist'n $q(y_i | y_{i-1})$ models the tag sequences
  - Emission dist'n $e(x_i | y_i)$ models words given their POS
- Q: How to we represent n-gram POS taggers?

Transitions and Emissions

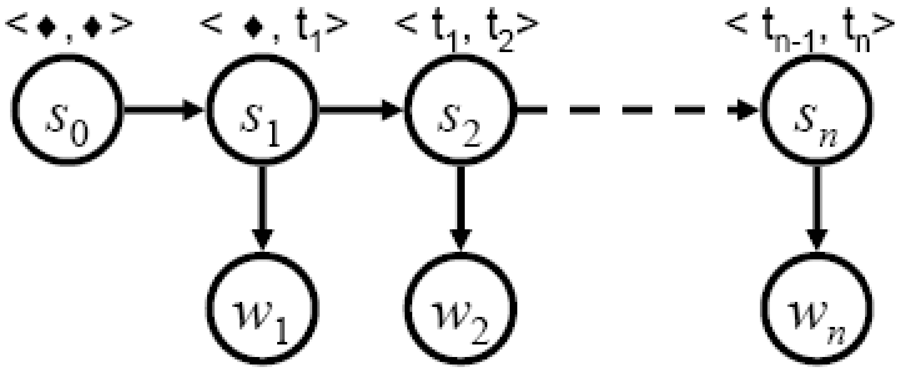


Transitions

- Transitions P(s|s') encode well-formed tag sequences
  - In a bigram tagger, states = tags



  - In a trigram tagger, states = tag pairs



Inference (Decoding)

- Problem: find the most likely (Viterbi) sequence under the model

$$\arg\max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$$

- Given model parameters, we can score any sequence pair

NNP	VBZ	NN	NNS	CD	NN	.
Fed	raises	interest	rates	0.5	percent	.

q(NNP|♦) e(Fed|NNP) q(VBZ|NNP) e(raises|VBZ) q(NN|VBZ)…..

- In principle, we’re done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

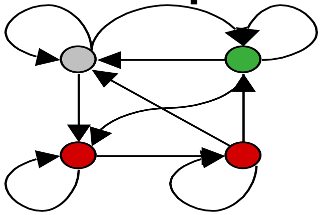
NNP VBZ NN NNS CD NN	⇨	logP = -23
NNP NNS NN NNS CD NN	⇨	logP = -29
NNP VBZ VB NNS CD NN	⇨	logP = -27

Entity Tagging with HMMs

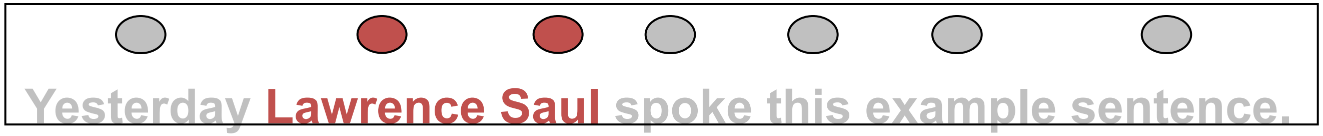
Given a sequence of observations:

Yesterday Lawrence Saul spoke this example sentence.

and a trained HMM:



Find the most likely state sequence: (Viterbi) $\arg\max_{\vec{s}} P(\vec{s},\vec{o})$



Yesterday **Lawrence Saul** spoke this example sentence.

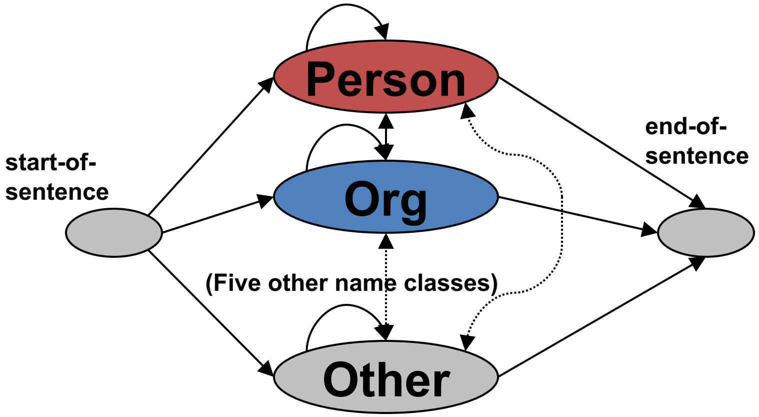
Any words said to be generated by the designated "person name" state extract as a person name:

Person name: Lawrence Saul

HMM Example: “Nymble”

[Bikel, et al 1998], [BBN “IdentiFinder”]

Task: Named Entity Extraction



Transition probabilities

$P(s_t \mid s_{t-1}, o_{t-1})$

Back-off to:

$P(s_t \mid s_{t-1})$

$P(s_t)$

Observation probabilities

$P(o_t \mid s_t, s_{t-1})$

or $P(o_t \mid s_t, o_{t-1})$

Back-off to:

$P(o_t \mid s_t)$

$P(o_t)$

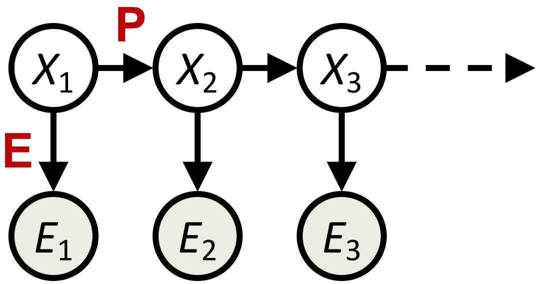
Train on 450k words of news wire text.

Results:

Case	Language	F1 .
Mixed	English	93%
Upper	English	91%
Mixed	Spanish	90%

Remaining HMM/BN Issues

- Where do **P**, **E** probabilities come from?
  - Training HMMs
  - Dealing with unobserved values



- What's the most likely explanation (a.k.a. decoding)
  - Needed for speech recognition, language parsing, …
  - Solution: Viterbi algoritm (most likely explanation)
  - What is we want multiple explanations? (beam search)

HMM Training: Degree of Supervision

- Supervised: Training data is tagged by humans
- Unsupervised: Training data isn’t tagged
- Partially supervised: Training data isn’t tagged, but you have a dictionary giving possible tags for each word

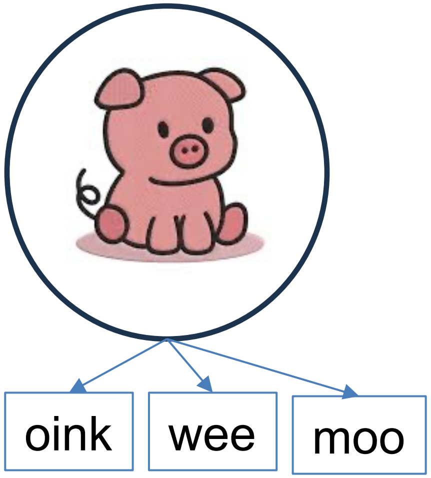
- We’ll start with the supervised case and move to decreasing levels of supervision

Training an HMM (Supervised)

1. Define model topology (states, possible arcs)
2. Obtain labeled/tagged data
3. Estimate HMM parameters:
    1. Transition probabilities
    2. Emission probabilities
4. Validate on hold-out data
5. Done.

MLE Example: Squeezy Pig

- Consider a random pig:
  - Observations:
    - squeeze it 10 times, and 8 times it goes "oink" and 2 times it goes “wee“
  - Model topology:
    - assume it is a stateless pig (no hidden states)
  - Parameter estimation:
    - Maximum likelihood estimate (MLE) to model the pig as a random emitter with P("oink") = 0.8 and P(“wee") = 0.2. P(“moo”)=?
    - P(x) = count(X) / total number of trials



oink

wee

moo

Learning: Maximum Likelihood

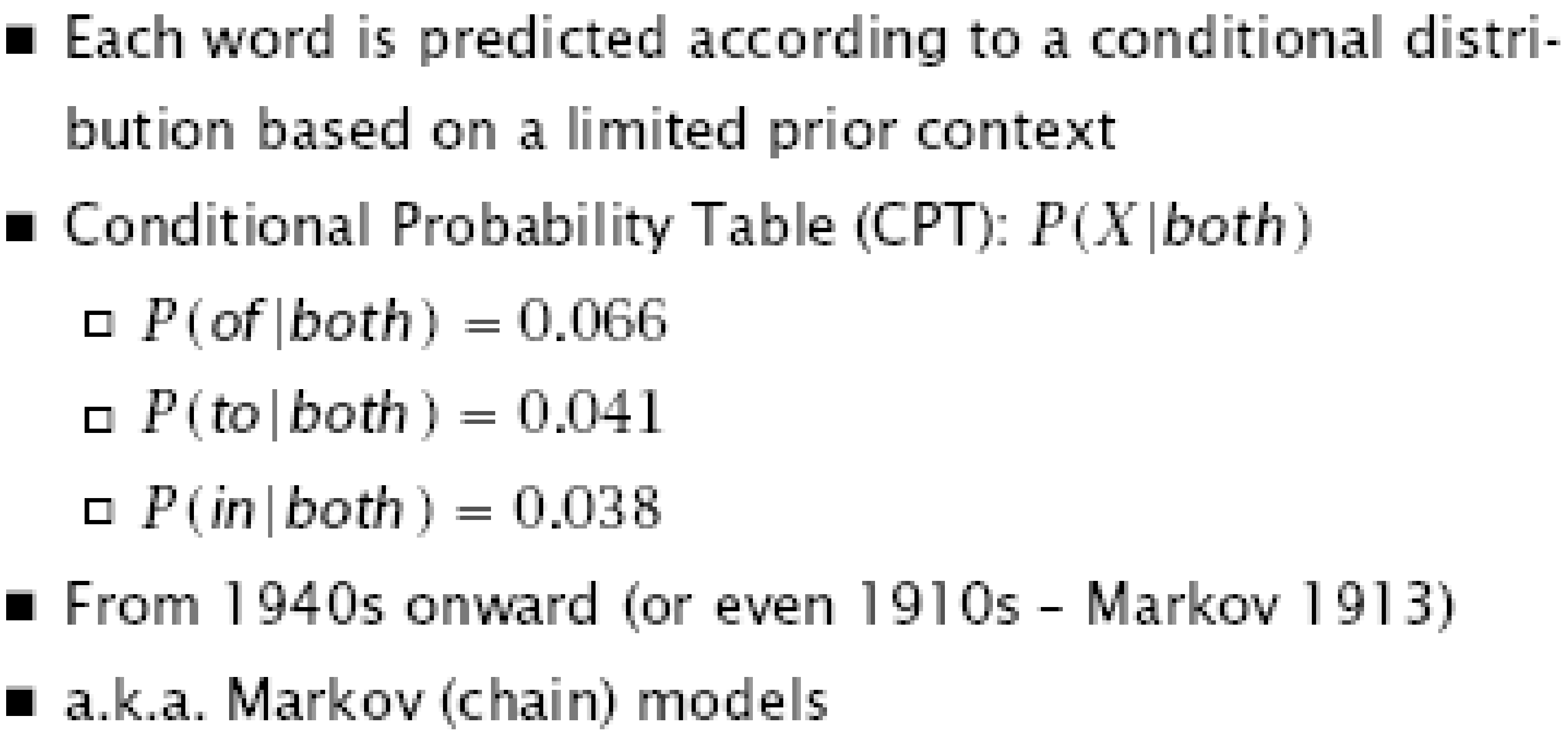
$$p(x_1 \ldots x_n, y_1 \ldots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- Learning
  - Maximum likelihood methods for estimating transitions q and emissions e

$$q_{ML}(y_i|y_{i-1}) = \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} \qquad e_{ML}(x|y) = \frac{c(y, x)}{c(y)}$$

  - Will these estimates be high quality?
    - Which is likely to be more sparse, q or e?

Estimating Transitions: N-gram model



- Each word is predicted according to a conditional distribution based on a limited prior context
- Conditional Probability Table (CPT): $P(X|both)$
  - □ $P(of|both) = 0.066$
  - □ $P(to|both) = 0.041$
  - □ $P(in|both) = 0.038$
- From 1940s onward (or even 1910s – Markov 1913)
- a.k.a. Markov (chain) models

Estimating Emission Probabilities

$$P(\mathbf{s}, \mathbf{w}) = \prod_i P(s_i|s_{i-1})P(w_i|s_i)$$

- Emissions are tricker:
  - Words we've never seen before
  - Will use new method: "smoothing"
  - One option: break out the Good-Turning smoothing
  - Issue: words aren't black boxes:

    343,127.23 11-year Minteria reintroducibly

  - Unknown words usually broken into word classes

    $D^+,D^+.D^+$ $D^+\text{-}x^+$ Xx^+ x^+"ly"

  - Another option: decompose words into features and use a maxent model along with Bayes' rule

$$P(w\,|\,t) = P_{MAXENT}(t\,|\,w)P(w)\,/\,P(t)$$

Problem: Insufficient data

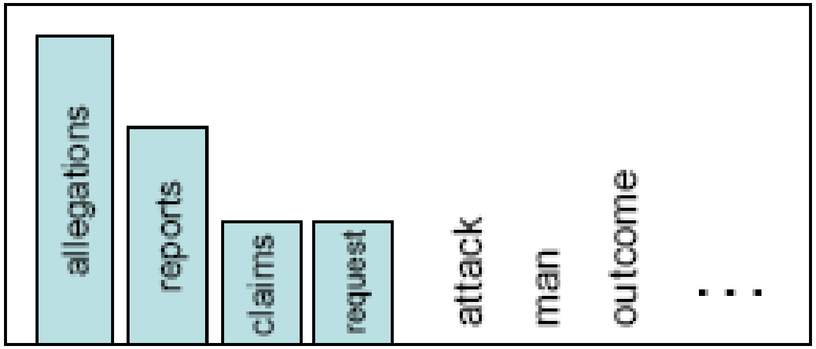
$$p(t \mid M_d) = 0$$

- Zero probability
  - May not wish to assign a probability of zero to an event that is never appeared in training data (unseen)
  - E.g., p("moo"| M_{pig})
- General approach
  - An unseen event is possible, but no more likely than would be expected by chance.
  - How to assign P(unseen)?

General Idea: Smoothing

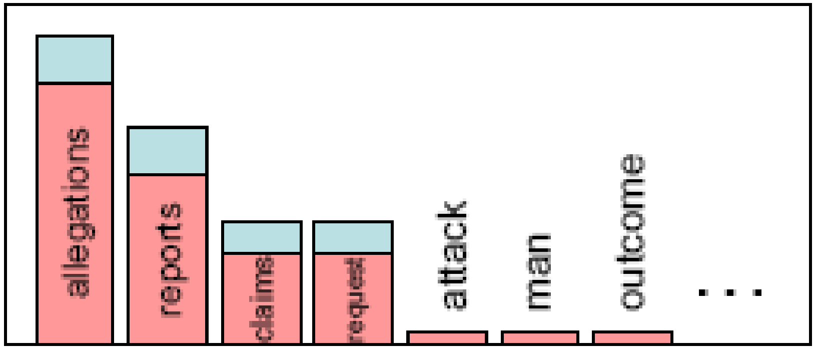
- We often want to make estimates from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



- Very important all over NLP, but easy to do badly!
- We'll illustrate with bigrams today (h = previous word, could be anything).

Example: Laplace smoothing

- Idea: pretend we saw every word once more than we actually did [Laplace]
  - Corresponds to a uniform prior over vocabulary
  - Think of it as taking items with observed count $r > 1$ and treating them as having count $r^* < r$
  - Holds out $V/(N+V)$ for “fake” events
  - $N1+/N$ of which is distributed back to seen words
  - $N0/(N+V)$ actually passed on to unseen words (nearly all!)
  - Actually tells us not only how much to hold out, but where to put it
- Addding 1 **poorly** in practice (why?)
- Quick fix: add some small δ instead of 1 [Lidstone-Jefferys]
- Better better, holds out less mass

HMMs: Degree of Supervision

- Supervised: Training corpus is tagged by humans
- Unsupervised: Training corpus isn’t tagged
- Partly supervised: Training corpus isn’t tagged, but you have a dictionary giving possible tags for each word

- We’ll start with the supervised case and move to decreasing levels of supervision

Motivation: Unsupervised Training

- Accuracy of tagging degrades outside of domain; Often make errors on important words (e.g., protein names)

$$\lambda = (A, B, \pi)$$

- We assumed that we know the underlying model
- Often these parameters are estimated on annotated training data, which has two drawbacks:
  - Annotation is difficult and/or expensive
  - Training data is different from the current data
- We want to maximize the parameters with respect to the current data, i.e., we're looking for a model λ', such that

$$\lambda' = \arg\max_{\lambda} P(O \mid \lambda)$$

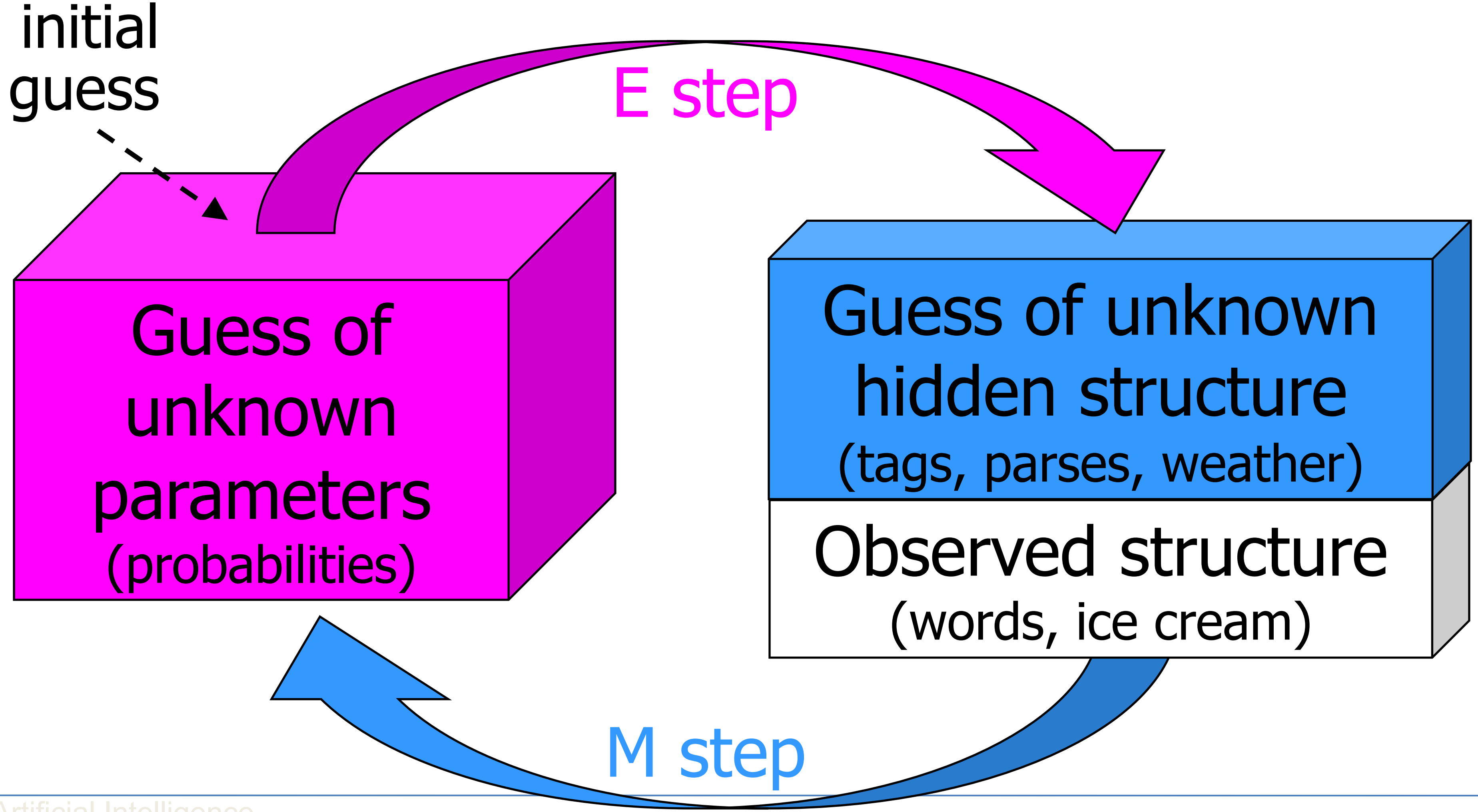
General Idea

- Start by devising a noisy channel
  - Any model that predicts the corpus observations via some hidden structure (tags, parses, pigs…)
- Initially **guess** the parameters of the model!
  - Educated guess is best, but random can work

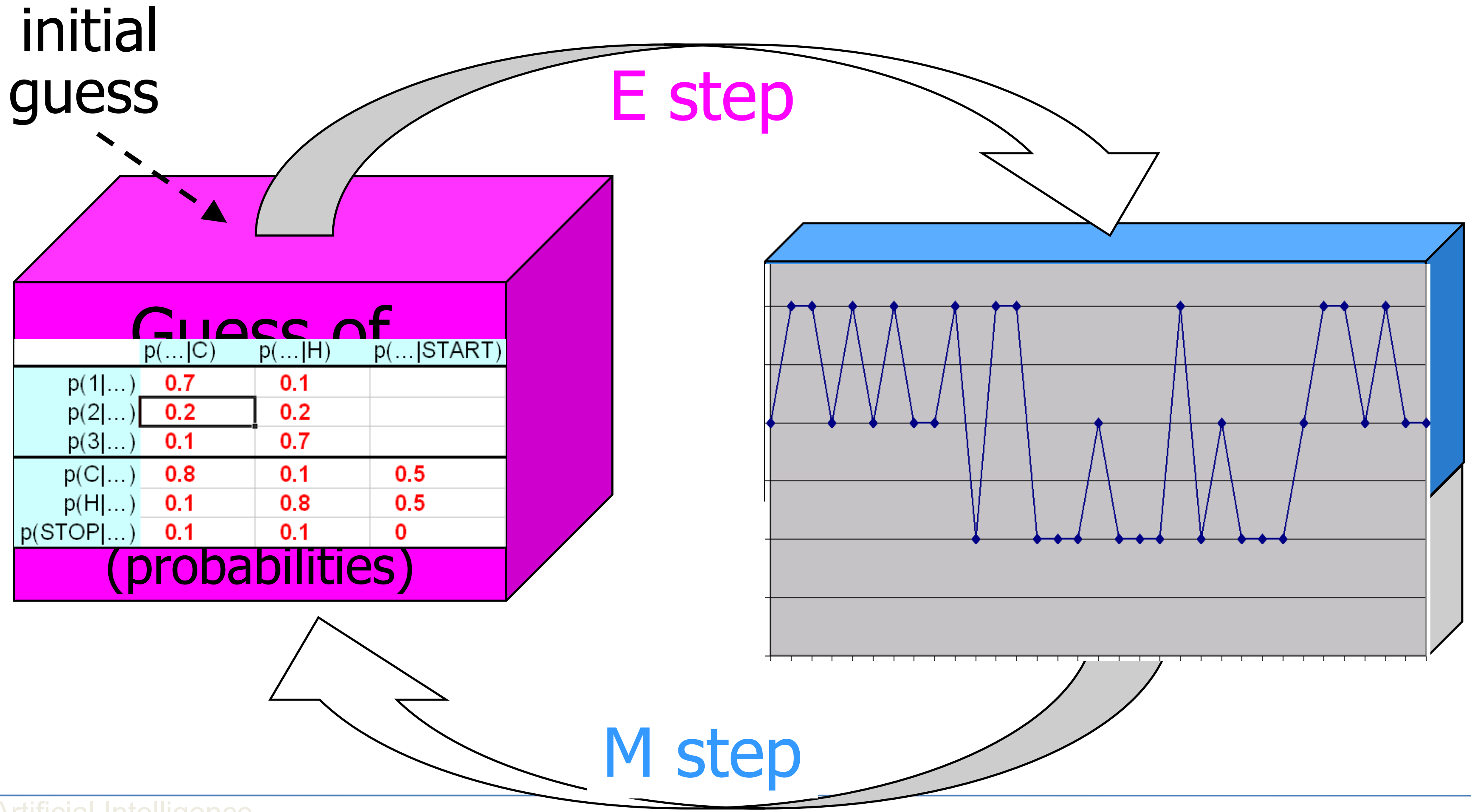
- **Expectation step:** Use current parameters (and observations) to reconstruct hidden structure
- **Maximization step:** Use that hidden structure (and observations) to reestimate parameters

Repeat until convergence!

General Idea


initial guess
E step
Guess of unknown parameters (probabilities)
Guess of unknown hidden structure (tags, parses, weather)
Observed structure (words, ice cream)
M step

For Hidden Markov Models



initial guess

E step

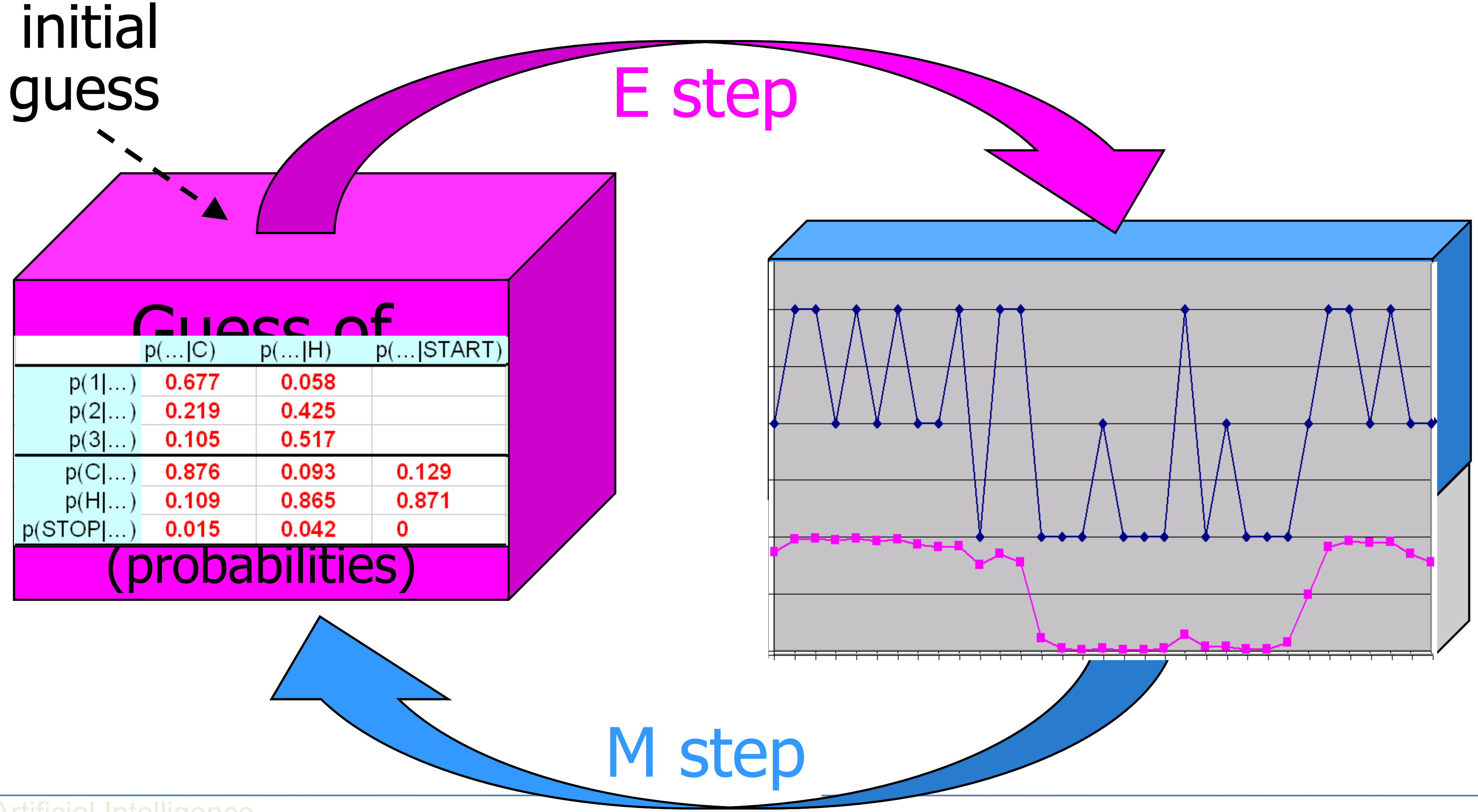
M step

Guess of

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.7	0.1	
p(2|…)	0.2	0.2	
p(3|…)	0.1	0.7	
p(C|…)	0.8	0.1	0.5
p(H|…)	0.1	0.8	0.5
p(STOP|…)	0.1	0.1	0

(probabilities)

For Hidden Markov Models


initial guess

Guess of

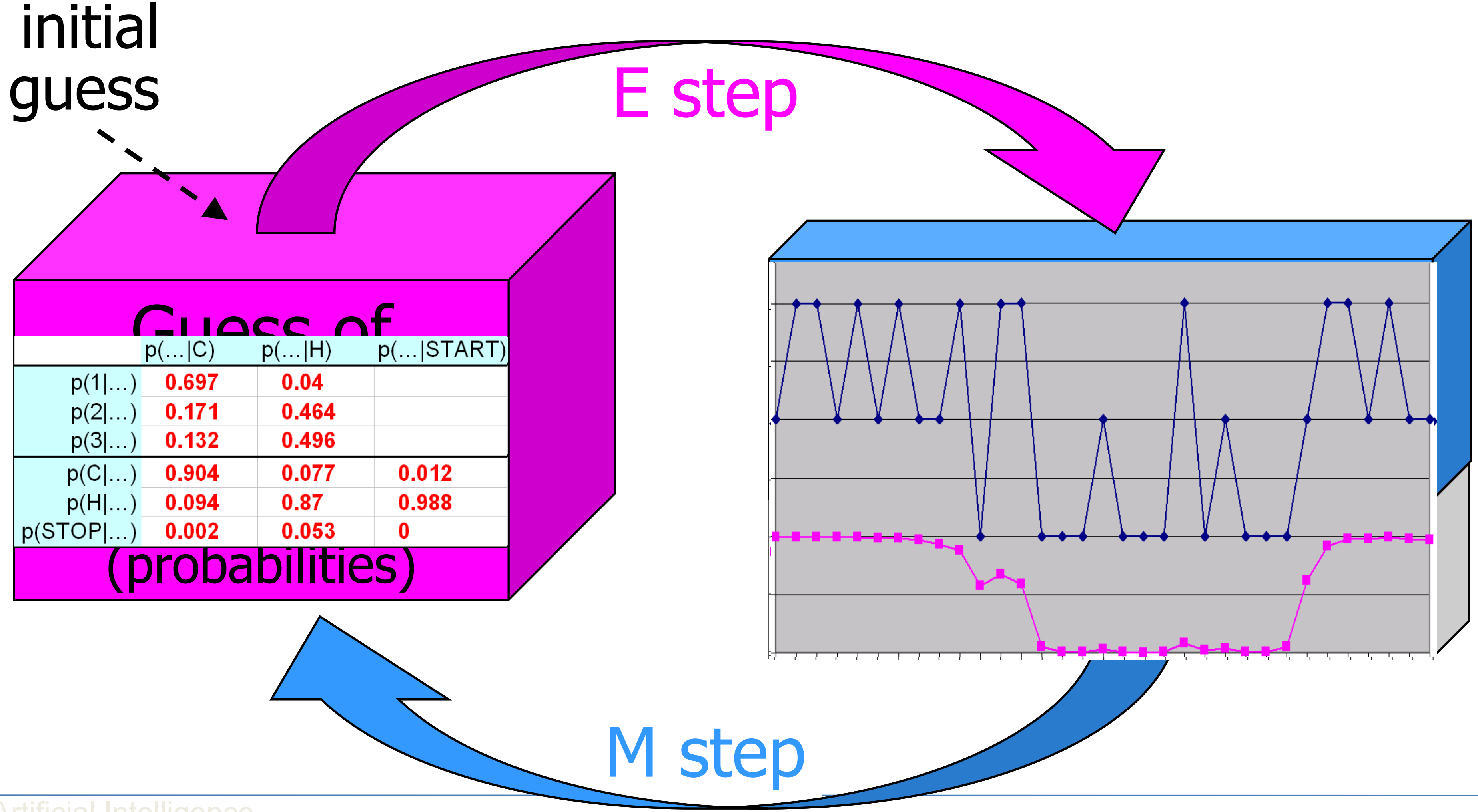
	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.677	0.058	
p(2|…)	0.219	0.425	
p(3|…)	0.105	0.517	
p(C|…)	0.876	0.093	0.129
p(H|…)	0.109	0.865	0.871
p(STOP|…)	0.015	0.042	0

(probabilities)

E step

M step

For Hidden Markov Models



	p(...|C)	p(...|H)	p(...|START)
p(1|...)	0.697	0.04	
p(2|...)	0.171	0.464	
p(3|...)	0.132	0.496	
p(C|...)	0.904	0.077	0.012
p(H|...)	0.094	0.87	0.988
p(STOP|...)	0.002	0.053	0

EM by Dynamic Programming: Two Versions

- The Viterbi approximation
  - Expectation: pick the *best* parse of each sentence
  - Maximization: retrain on this best-parsed corpus
  - *Advantage:* Speed!

- Real EM

why slower?

  - Expectation: find *all* parses of each sentence
  - Maximization: retrain on *all* parses in proportion to their probability (as if we observed fractional count)
  - *Advantage:* p(training corpus) guaranteed to increase
  - Exponentially many parses, so don't extract them from chart – need some kind of clever counting

Baum-Welch (1)

- Want to estimate:
  - a_{ij} (transition probability from state i to state j)
- If we had tagged corpus:
  - a_{ij} = count(transition from state i to state j) / count(transition from state i to any state)
- Don't have one. So:
  - **Assume some initial values** for a_{ij} and $b_j(w)$
  - Use a_{ij} and $b_j(w)$ to compute expected counts (*E*)
  - Then use *E* to *re-estimate* $a_{ij} \rightarrow a'_{ij}$
  - a'_{ij} = *E*(transition from state i to state j) / *E*(transition from state i to any state)

BM: Intuition

- If training data has information about sequence of hidden states (as in word recognition example), then use maximum likelihood estimation of parameters:

$$a_{ij} = P(s_i \mid s_j) = \frac{\text{Number of transitions from state } S_j \text{ to state } S_i}{\text{Number of transitions out of state } S_j}$$

$$b_i(v_m) = P(v_m \mid s_i) = \frac{\text{Number of times observation } V_m \text{ occurs in state } S_i}{\text{Number of times in state } S_i}$$

BM: Approximation

General idea:

$$a_{ij} = P(s_i \mid s_j) = \frac{\text{Expected number of transitions from state } S_j \text{ to state } S_i}{\text{Expected number of transitions out of state } S_j}$$

$$b_i(v_m) = P(v_m \mid s_i) = \frac{\text{Expected number of times observation } V_m \text{ occurs in state } S_i}{\text{Expected number of times in state } S_i}$$

$\pi_i = P(s_i) =$ Expected frequency in state S_i at time $k=1$.

BM Algorithm: Iterations

- Recurrence:
  1. Estimate $A_{k,l}$ and $E_k(b)$ from $a_{k,l}$ and $e_k(b)$ overall all training sequences (E-step)
  2. Update $a_{k,l}$ and $e_k(b)$ using ML (M-step)
  3. Repeat steps #1, #2 with new parameters $a_{k,l}$ and $e_k(b)$

- Initialization:
  - Set A and E to pseudocounts (or priors)

- Termination: if *Δlog-likelihood < threshold* or Ntimes>max_times

BM Algorithm: Iterations (2)

- Recurrence:

1. Calculate forward/backwards probs, $f_k(i)$ and $b_k(i)$, for each training sequence

2. E-step: Estimate the expected number of k→l transitions, $A_{k,l}$

$$A_{k,l} = \sum_i f_k(i) \cdot a_{k,l} \cdot e_l(x_{i+1}) \cdot b_l(i+1) / P(\vec{x} \mid \theta)$$

and the expected number of symbol b appearences in state k, $E_k(b)$

$$E_k(b) = \sum_{\{i \mid x_i = b\}} f_k(i) \cdot b_k(i) / P(\vec{x} \mid \theta)$$

3. M-step: Estimate new model parameters $a_{k,l}$ and $e_k(b)$ using ML across all training sequences

4. Estimate the new model's (log)likelihood to assess convergence

An example of Baum-Welch

(thanks to Sarah Wheelan, JHU)

- I observe the dog across the street. Sometimes he is inside, sometimes outside.
- I assume that since he can not open the door himself, then there is another factor, hidden from me, that determines his behavior.
- Since I am lazy, I will guess there are only two hidden states, S_1 and S_2.

An example of Baum-Welch (cntd)

- Estimating initial probabilities:
  - Assume all sequences start with hidden state S_1, calculate best probability
  - Assume all sequences start with hidden state S_2, calculate best probability
  - Normalize to 1.

- Now, we have generated the updated transition, emission and initial probabilities. Repeat this method until those probabilities converge

An example of Baum-Welch (cntd)

- One set of observations:
  - I-I-I-I-I-O-O-I-I-I
- Guessing two hidden states. I need to invent a transition and emission matrix.
  - Note: since Baum-Welch is an EM algorithm the better my initial guesses are the better the job I will do in estimating the true parameters

		Day *k+1*	
		S1	S2
Day *k*	S1	0.5	0.5
	S2	0.4	0.6

	IN	OUT
S1	0.2	0.8
S2	0.9	0.1

An example of Baum-Welch (cntd)

- Let's assume initial values of:
  - $P(S_1) = 0.3$, $P(S_2) = 0.7$

- Example guess: if initial I-I came from S_1-S_2 then the probability is:
  $0.3 \times 0.2 \times 0.5 \times 0.9 = 0.027$

Day *k+1*

Day *k*

	S1	S2
S1	0.5	0.5
S2	0.4	0.6

	IN	OUT
S1	0.2	0.8
S2	0.9	0.1

An example of Baum-Welch (cntd)

- Now, let's estimate the transition matrix. Sequence I-I-I-I-I-O-O-I-I-I has the following events:
    - II, II, II, II, IO, OO, OI, II, II
- So, our estimate for S_1->S_2 transition probability is:
    - 0.285/2.4474 = 0.116
- Similarly, calculate the other three transition probs and normalize so they sum up to 1
- Update transition matrix

Seq	P(Seq) for S_1S_2	Best P(Seq)
II	0.027	0.3403 S_2S_2
II	0.027	0.3403 S_2S_2
II	0.027	0.3403 S_2S_2
II	0.027	0.3403 S_2S_2
IO	0.003	0.2016 S_2S_1
OO	0.012	0.0960 S_1S_1
OI	0.108	0.1080 S_1S_2
II	0.027	0.3403 S_2S_2
II	0.027	0.3403 S_2S_2
Total	**0.285**	**2.4474**

Summary – Baum-Welch Algorithm

- Start with initial HMM
- Calculate, using F-B, the likelihood to get our observations given that a certain hidden state was used at time i.
- Re-estimate the HMM parameters
- Continue until convergence
- Baum showed this to monotonically improve data likelihood (not necessarily accuracy!)
- Typically requires MANY observations, and (relatively) few parameters → simple topology

Summary: HMMs

- Widely used in AI: robotics, speech, NLP, …
- Often have large number of hidden states (e.g., words in vocabulary → 500K). Optimization needed
- Training exhaustively using MLE not feasible, need to generalize to unseen input (smoothing, classes)

- Now: **Markov Decision Process**.
  - How to make sequential decisions to optimize total expected future rewards

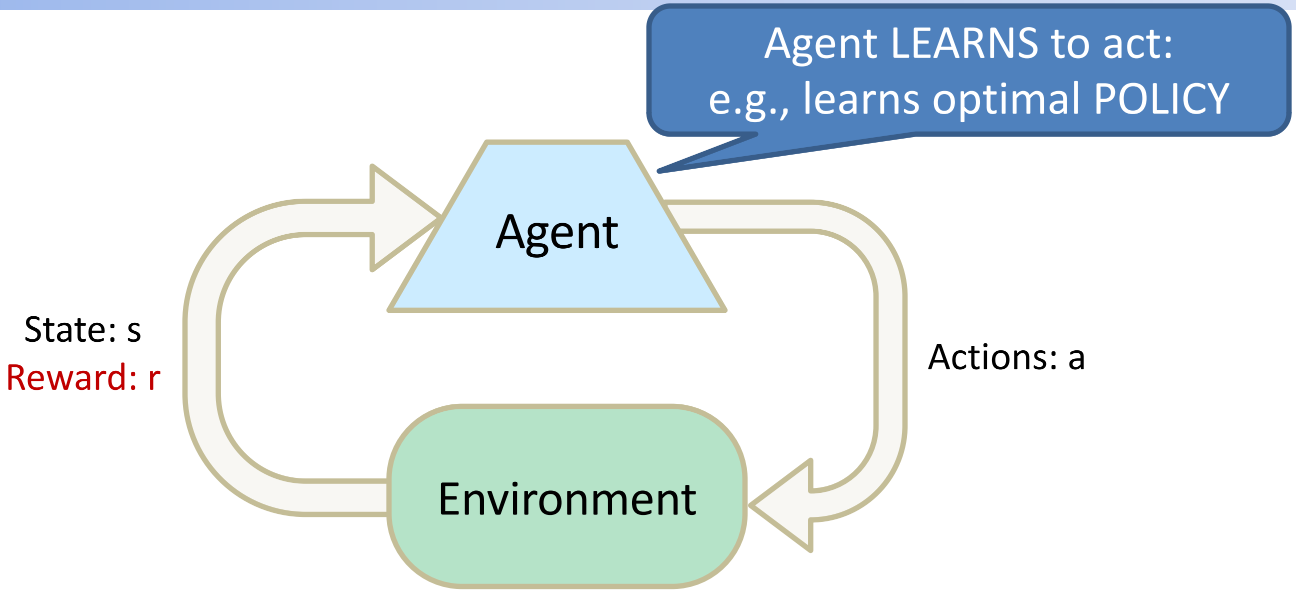
Big Picture

- AI as Planning:
  - ✓ Model of the world known (utilities, action outcomes)
  - ✓ Deterministic search: UCS, A*, MiniMax
  - ✓ Non-deterministic search → ExpectiMax
  - ➢ Inference under uncertainty: BNs, HMMs
- ➢ AI as Learning:
  - – Model of world *partially* known (rewards? outcomes?)

    → Markov Decision Processes (Today)
  - – Rewards, action outcomes unknown

    → Reinforcement Learning (After Spring Break)

Big Picture: Learning Agents



- 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!

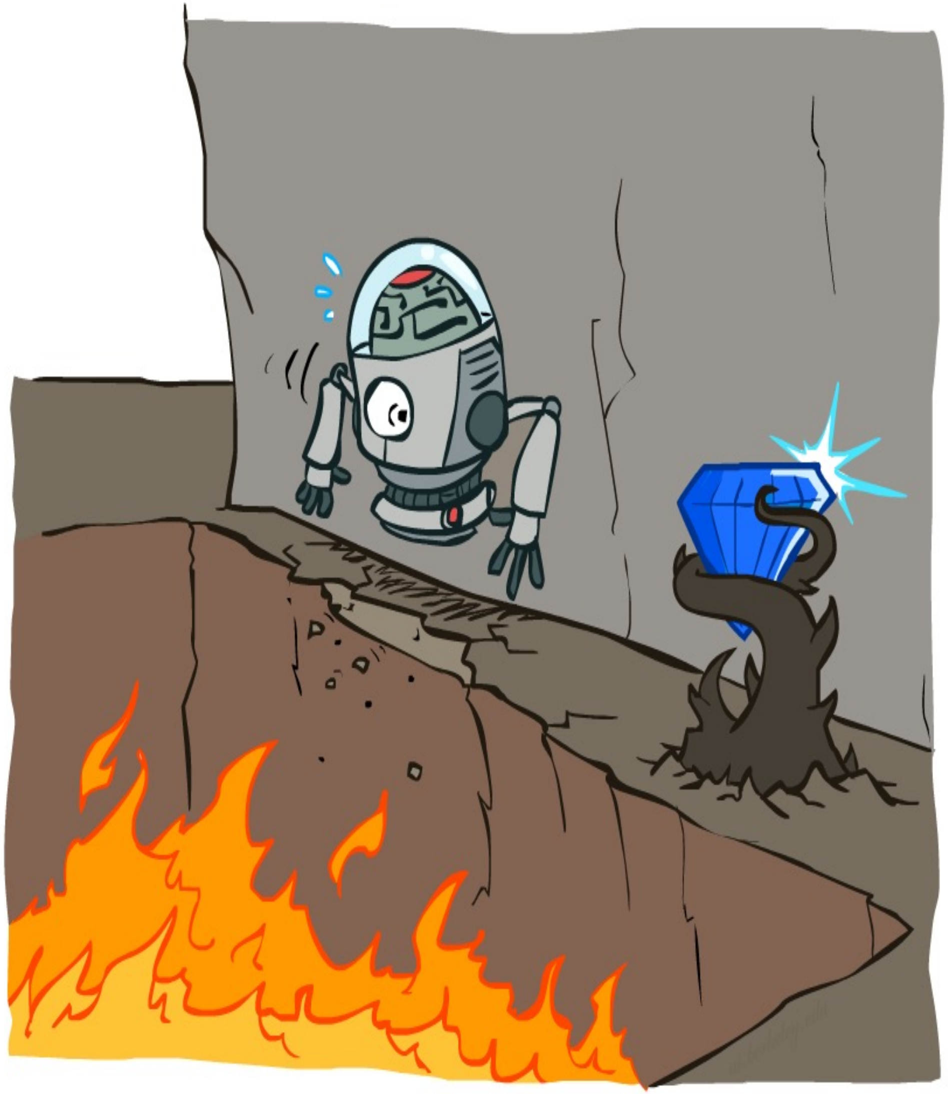
Rewards, Transitions known: MDP

Rewards, Transitions not known: ????

Plan (Next few weeks)

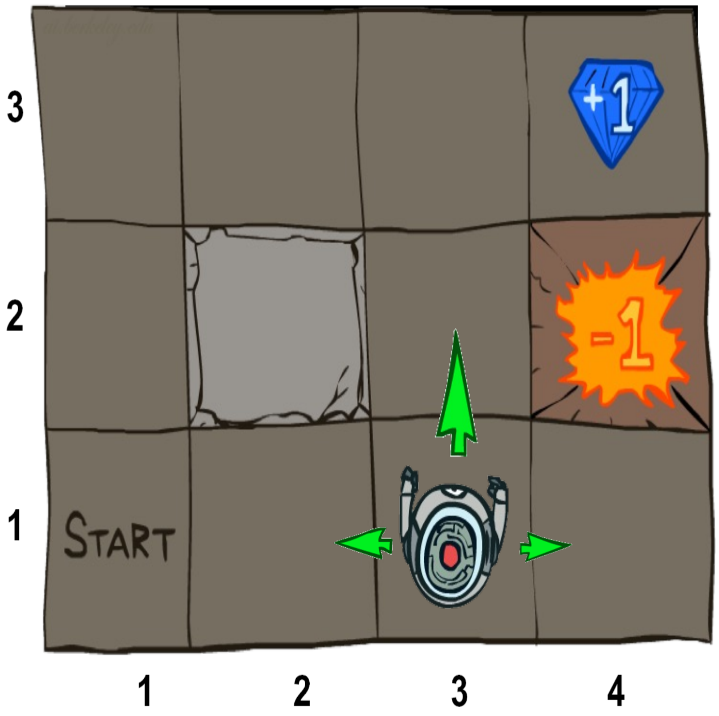
- Markov Decision Processes (MDPs)
  - MDP formalism
  - Solution: Value Iteration and Policy Iteration
- Reinforcement Learning (RL)
  - Relationship to MDPs
  - Several learning algorithms
  - RL applications to games, “real world”
- Project 3: MDPs, RL for Pacman and gridworld

Non-Deterministic **Actions**



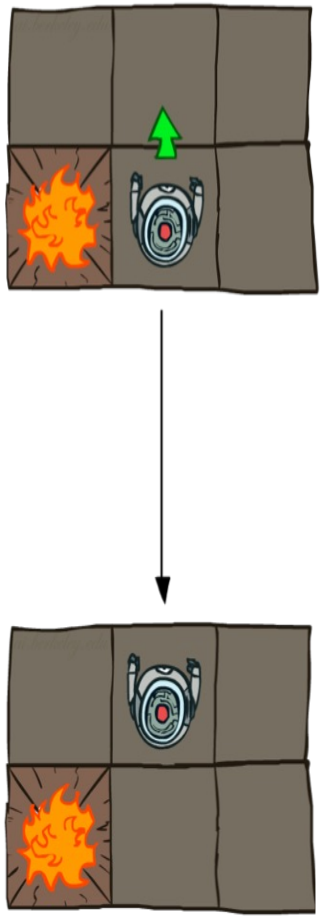
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- **Noisy movement**: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize the sum of rewards

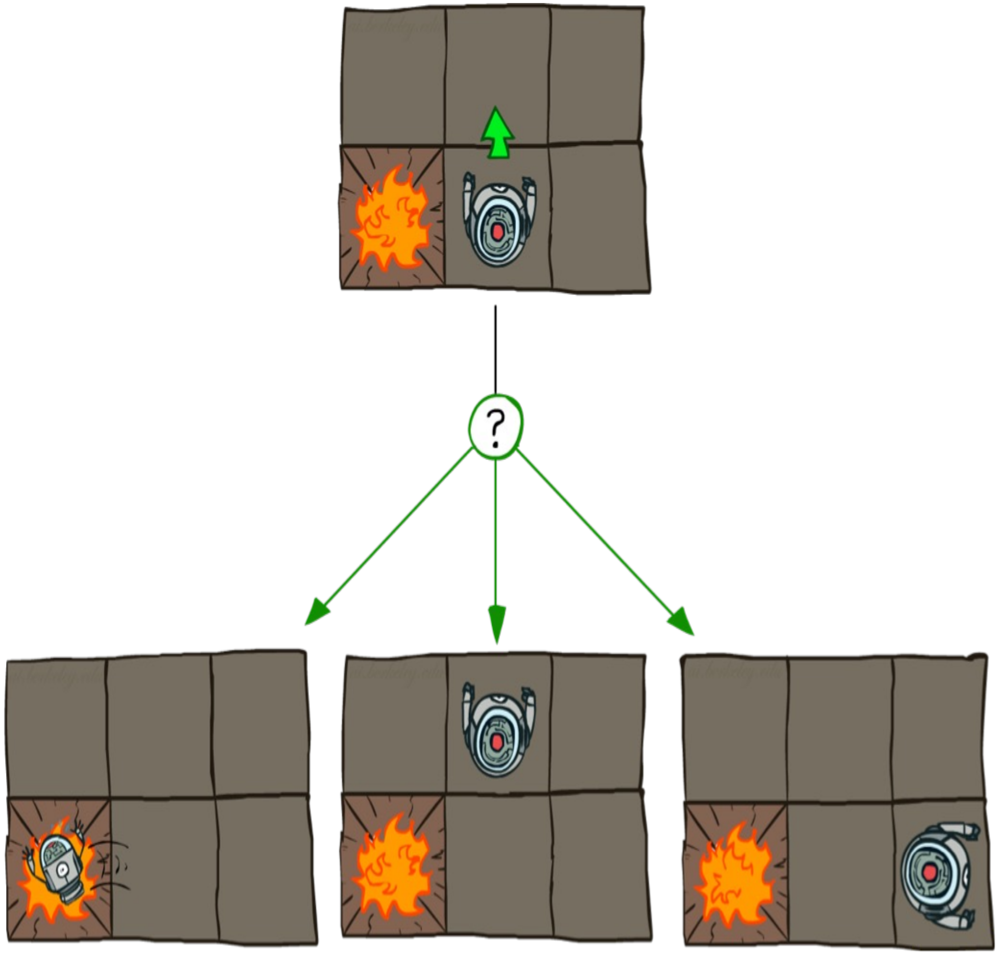


Grid World Actions

Deterministic Grid World

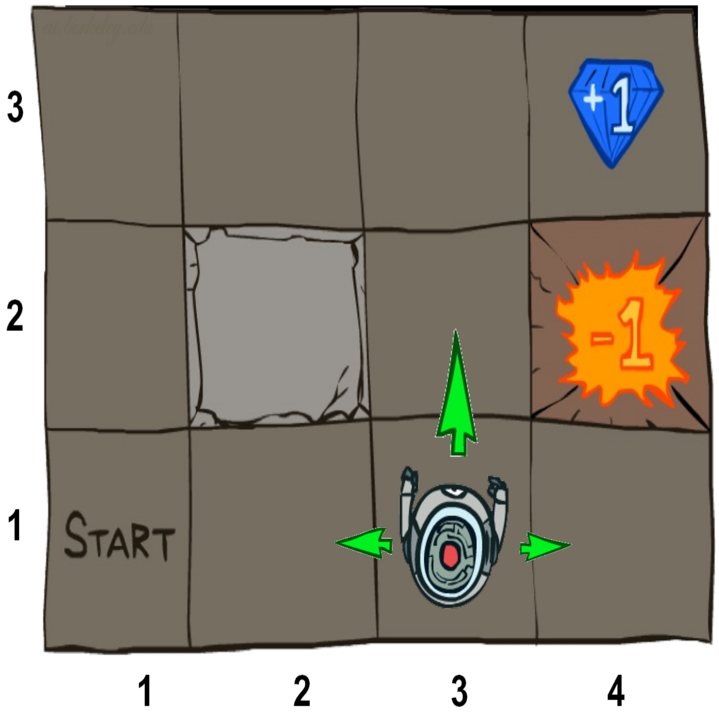


Stochastic Grid World



New Idea: Markov Decision Process (MDP)

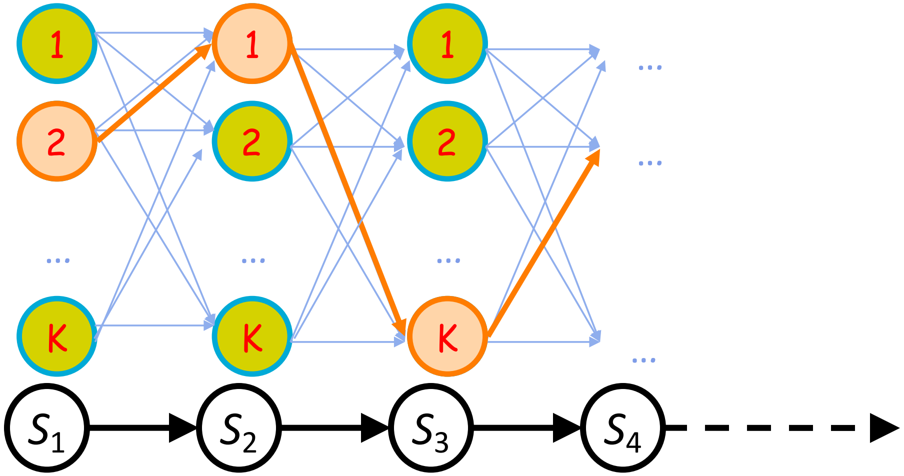
- An MDP is defined by:
  - Set of states $s \in S$
  - Actions $a \in A$
  - Transition function $T(s, a, s')$
    - Probability that a from s leads to s', i.e., $P(s' | s, a)$
    - Also called the model or the dynamics
  - Reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - Start state (s_0)
  - Terminal state (optional)



- MDPs are non-deterministic search problems
  - One way to solve them is with **expectimax** search
  - We can do better

What is “Markov” about MDPs?

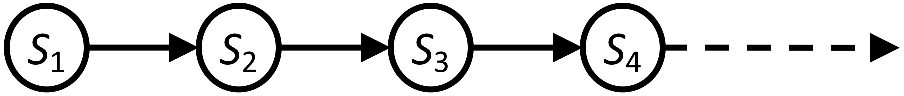
- Remember: “Markov” means that given the current state, the future and the past are independent



Andrey Markov
(1856-1922)

- Like (H) MMs, where the successor function only depends on current state (not the full history)

What is "Markov" about MDPs (2)?



- Markov decision processes, "Markov" means action outcomes depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

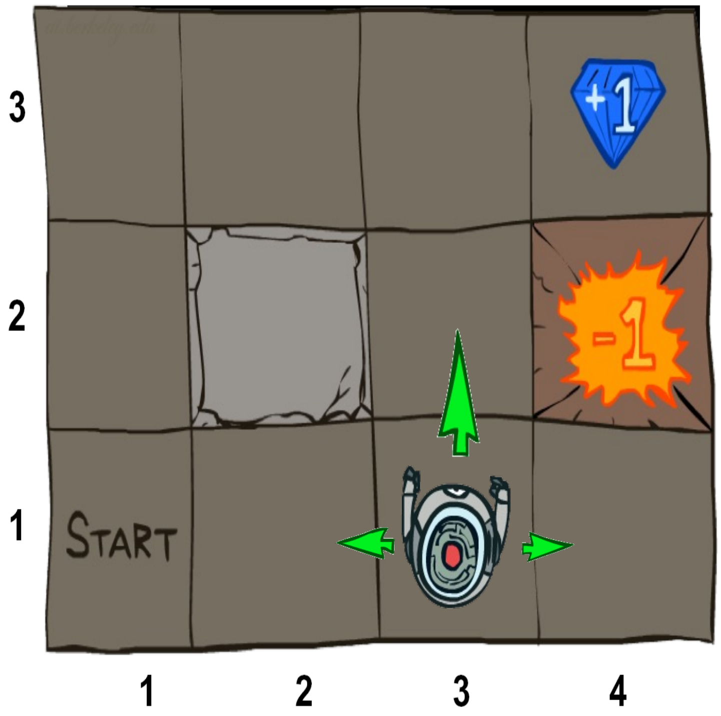
$$=$$

$$\boxed{P(S_{t+1} = s' | S_t = s_t, A_t = a_t)}$$

- Like (H) MMs, where the successor function only depends on current state (not the full history)

New Idea: Markov Decision Process (MDP)

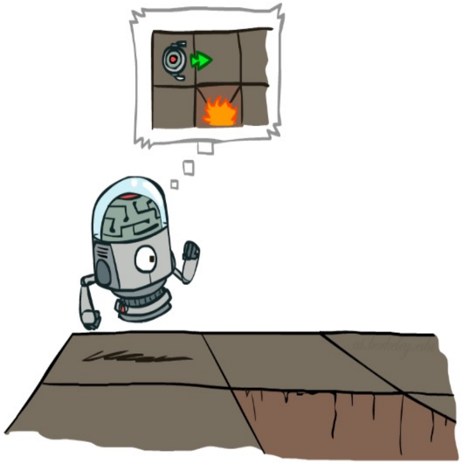
- An MDP is defined by:
  - Set of states s ∈ S
  - Actions a ∈ A
  - Transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics
  - Reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - Start state (s_0)
  - Terminal state (optional)

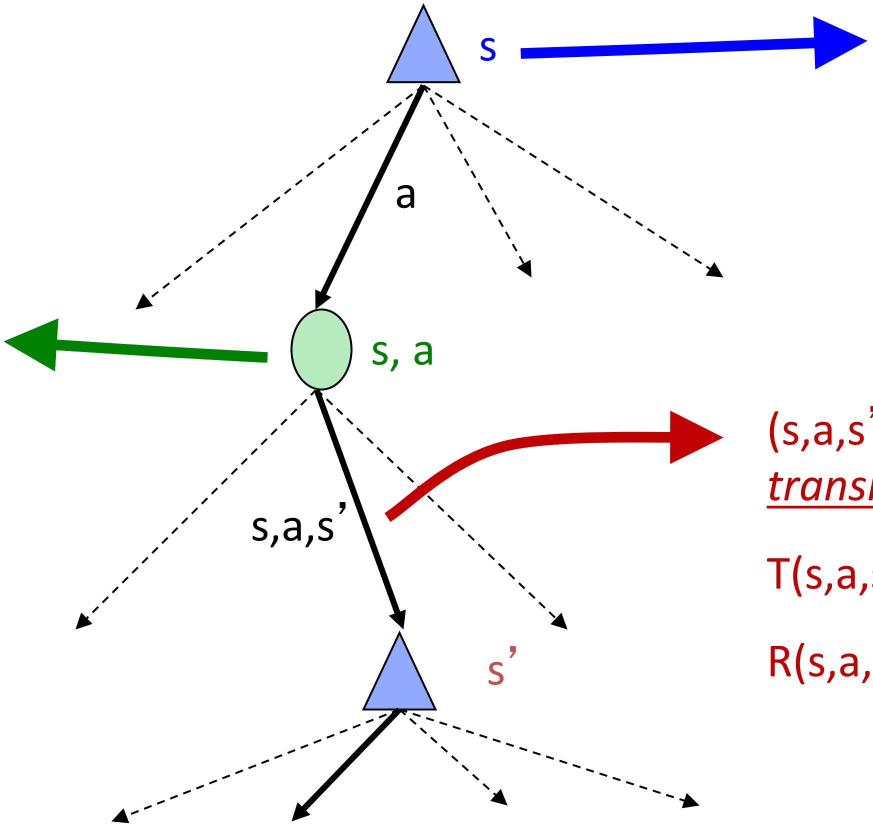


- MDPs are non-deterministic search problems
  - One way to solve them is with **expectimax** search
  - We can do better

MDP Search Trees

- Each MDP state can be viewed as an expectimax search node





s

a

s, a

s,a,s'

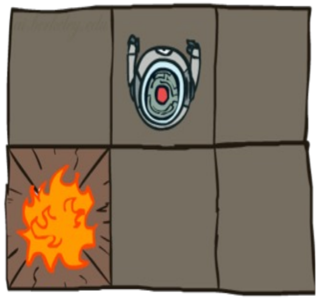
s'

s is a *state*

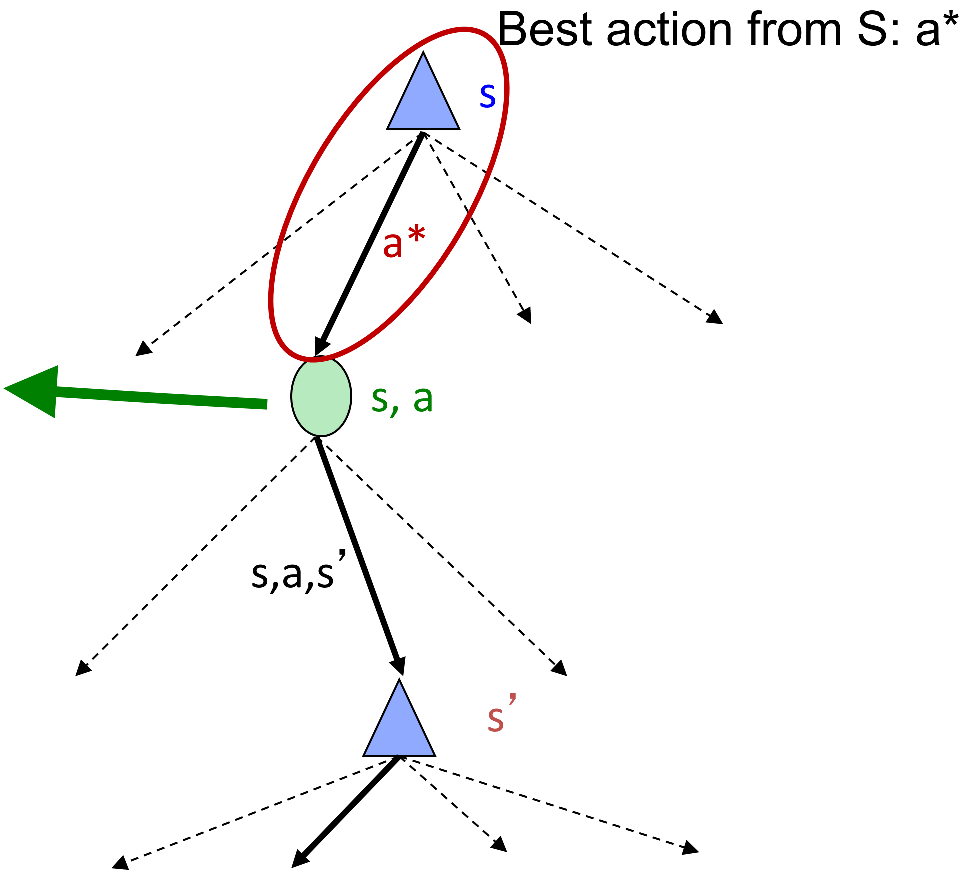
(s,a,s') called a *transition*

T(s,a,s') = P(s'|s,a)

R(s,a,s')



Expectimax Solution: Best Action from S

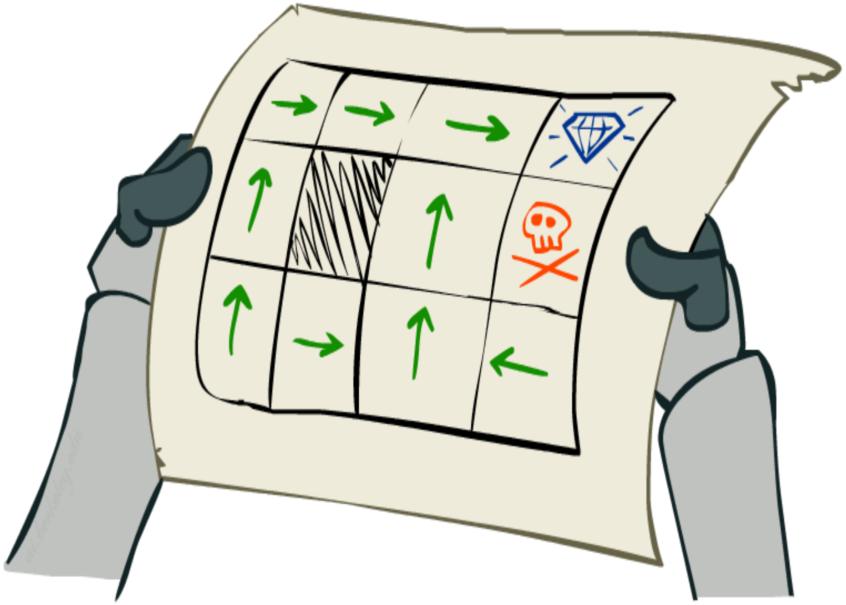


Definition: **Policy**

- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

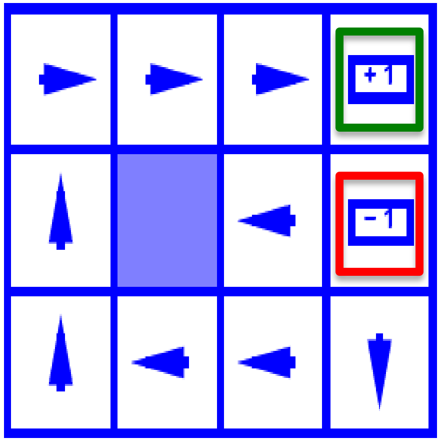
- Minimax/Expectimax do NOT compute entire policies
  - **They computed the action for a single state only (and then re-plan)**



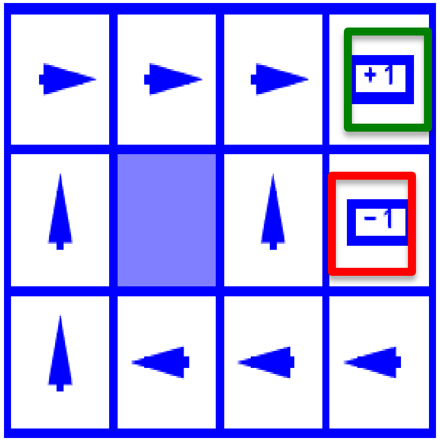
Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

Optimal Policies

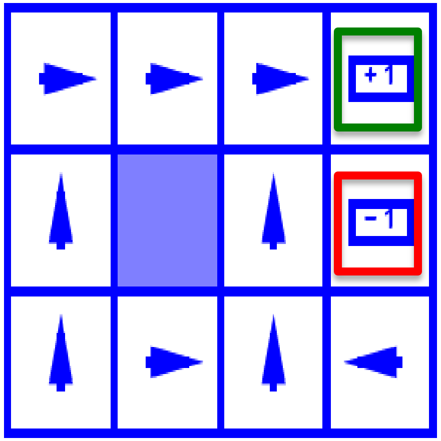
R(s) = the “living reward”



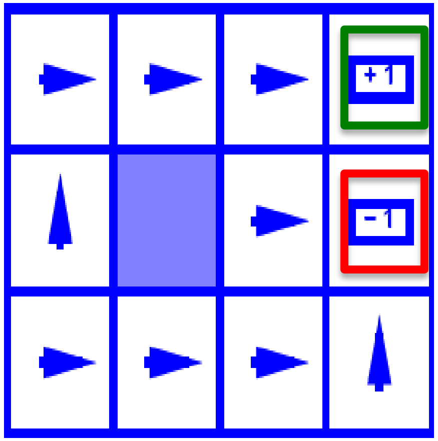
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Utilities of Sequences

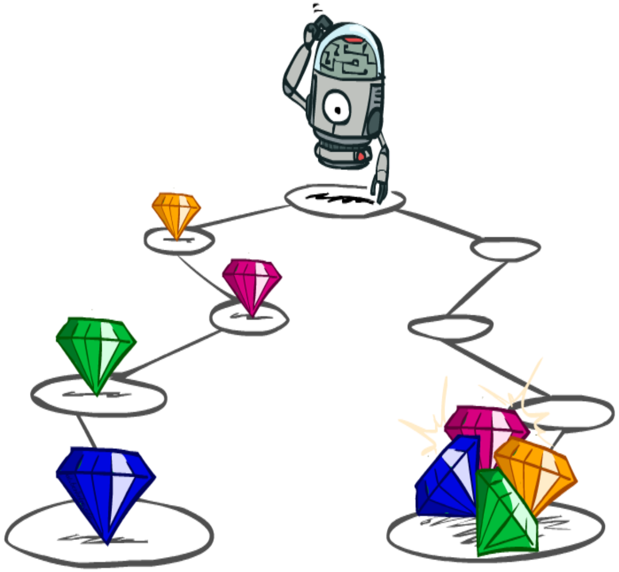
- What preferences should an agent have over reward sequences?

[1, 2, 2] or [2, 3, 4]

- More or less?

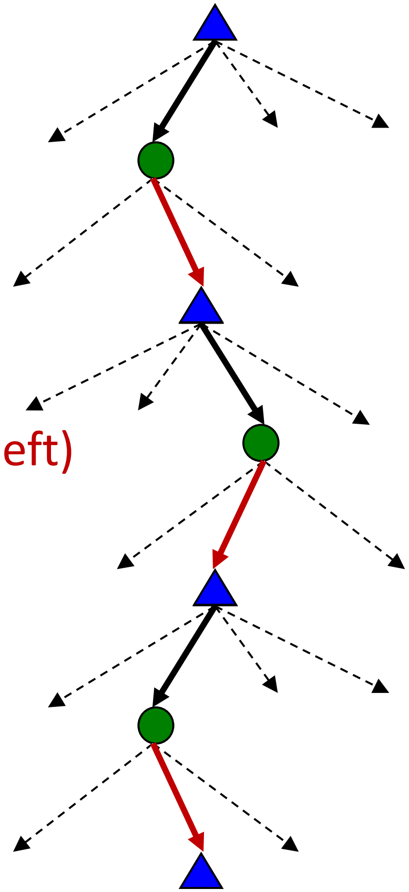
[0, 0, 1] or [1, 0, 0]

- Now or later?



Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?



- Solutions:
  1. Finite horizon: (similar to depth-limited search)
     - Terminate episodes after a fixed T steps (e.g. assume finite lifetime)
     - *Problem*: Gives nonstationary policies (π depends on time left)
  2. Discounting: use 0 < γ < 1

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

     - Smaller γ → smaller "horizon" – shorter term focus
  3. Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

Discounting

- It’s reasonable to maximize the sum of rewards
- It’s also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

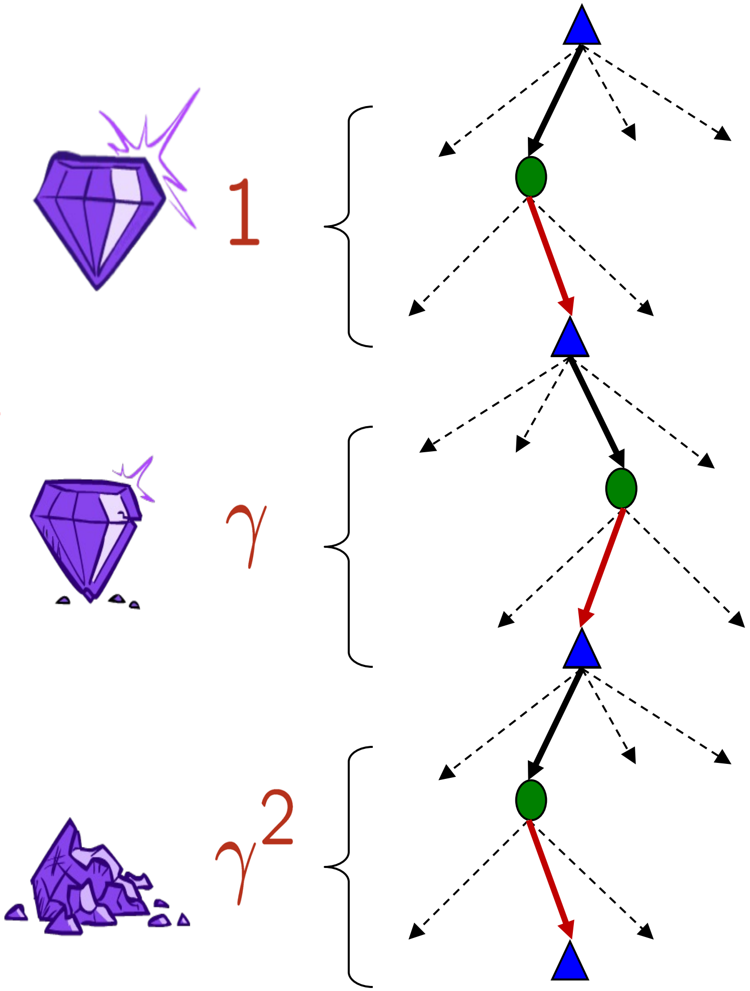
Worth Next Step

γ^2

Worth In Two Steps

Big Idea: Reward Discounting

- How to discount?
  - Each time we descend a level, we multiply by the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



Detour: Temporal/Delay Discounting

- What would you rather have?
  - A. $100 today
  - B. $150 a year from now
- What about:
  - A. $100 in 12 months
  - B. $110 in 13 months
- Humans temporally discount values of rewards
  - https://en.wikipedia.org/wiki/Temporal_discounting
- Delayed gratification: https://www.youtube.com/watch?v=QX_oy9614HQ

Quiz: Discounting

- Given:

10				1
a	b	c	d	e

  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic (no noise, for now)

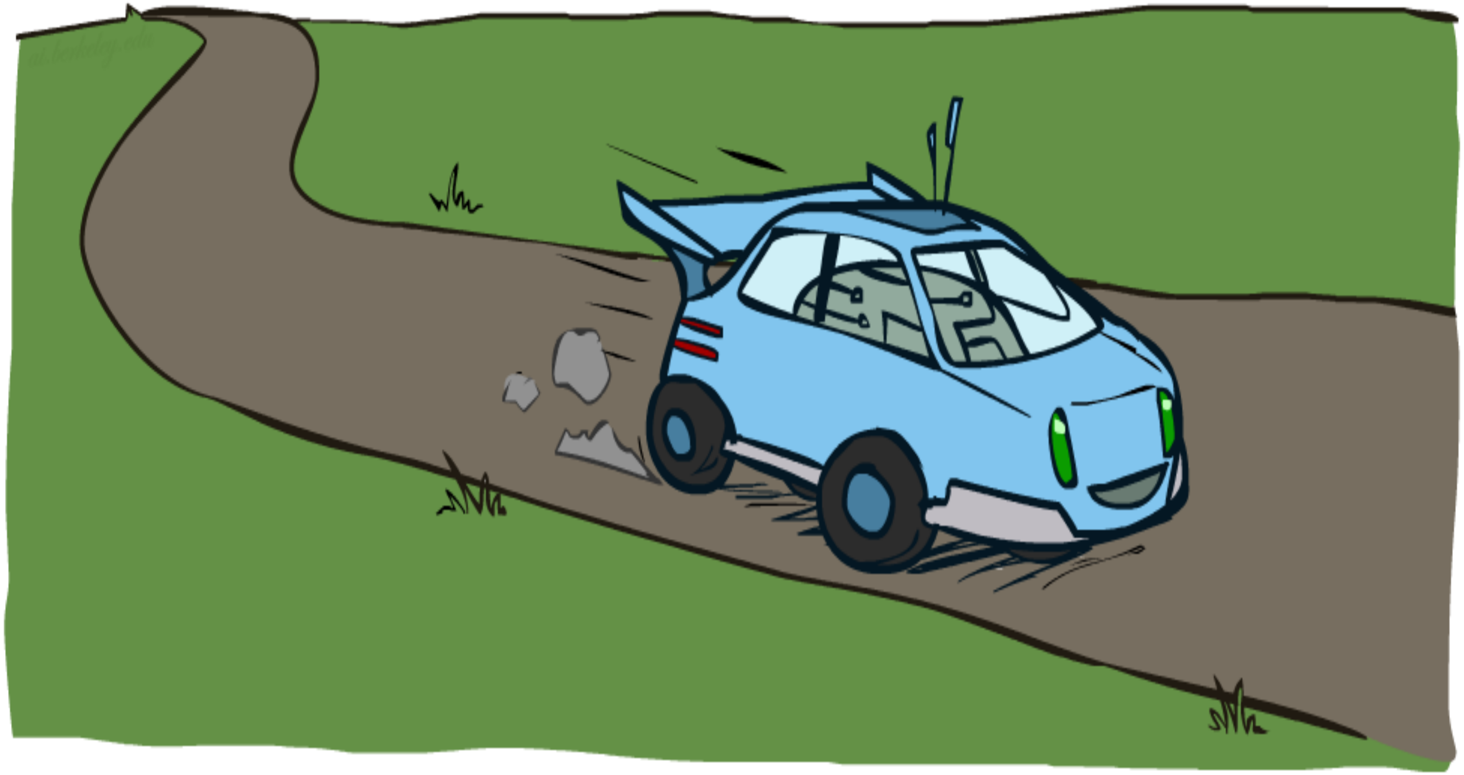
- P 1: For $\gamma = 1$, what is the optimal policy?

- P 2: For $\gamma = 0.1$, what is the optimal policy?

- P 3: For which γ are West and East equally good when in state d?

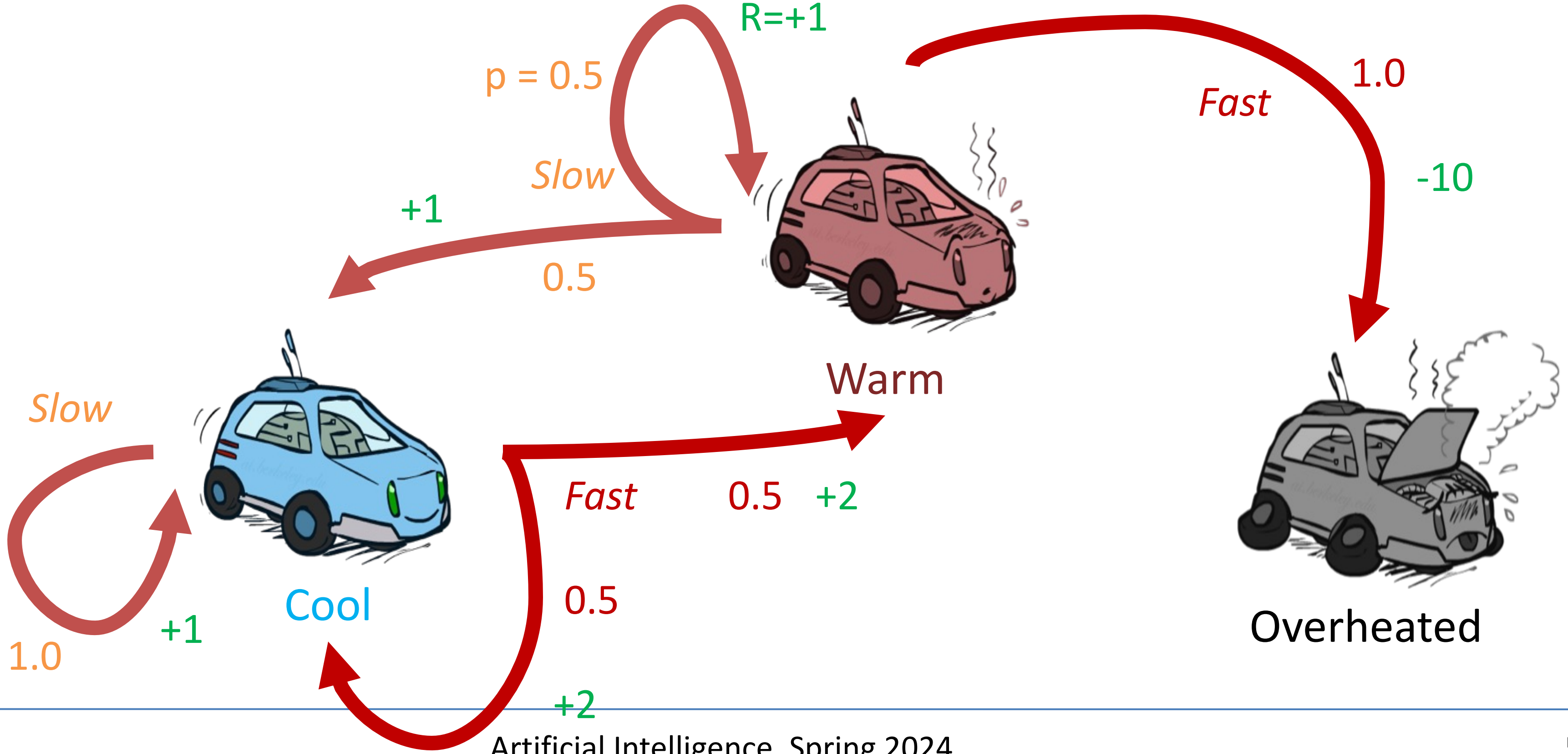
https://www.wolframalpha.com/input/?i=10x%5E3+%3D+x

Example: Car Racing

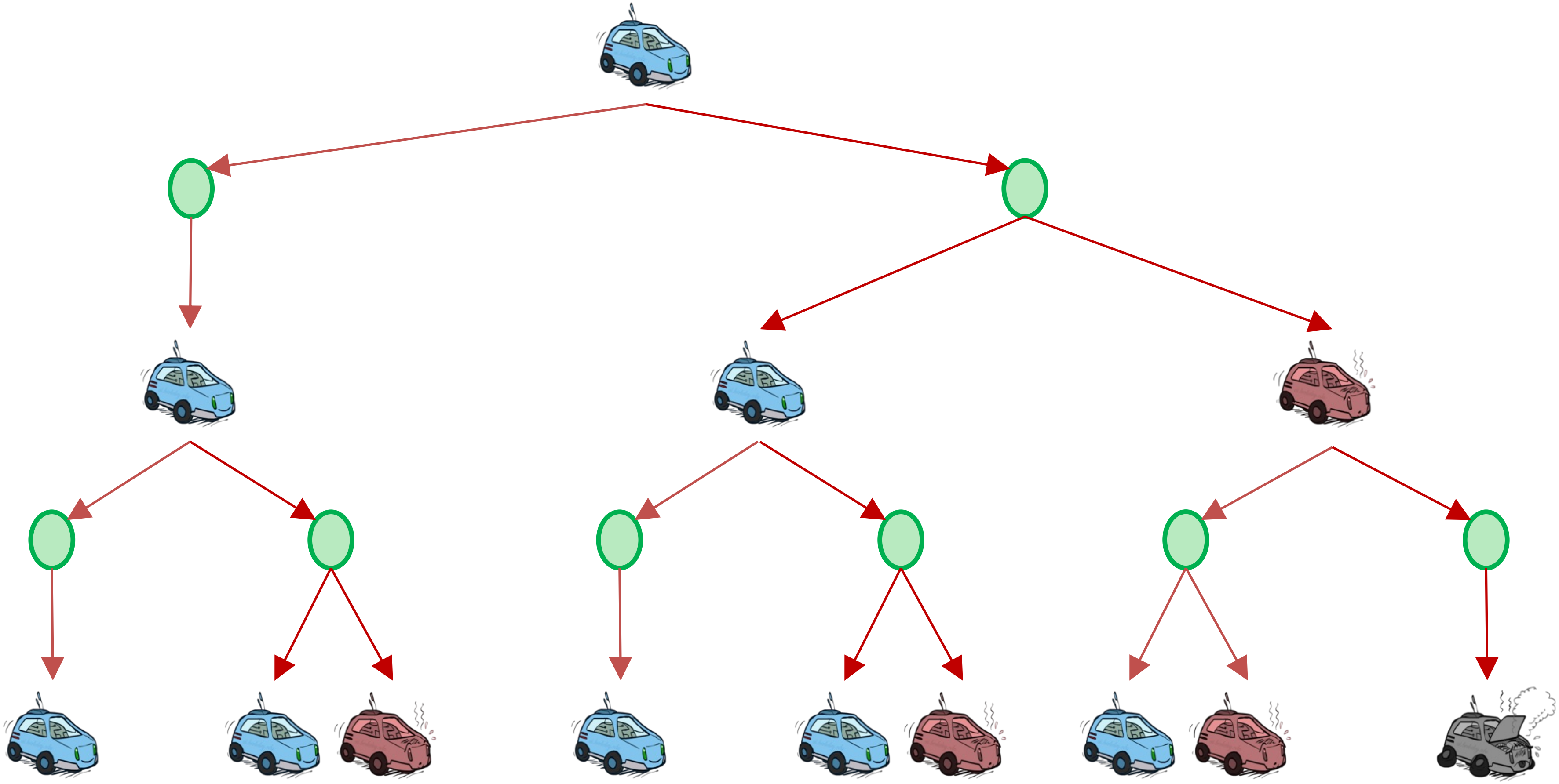


Car Racing: State Diagram

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



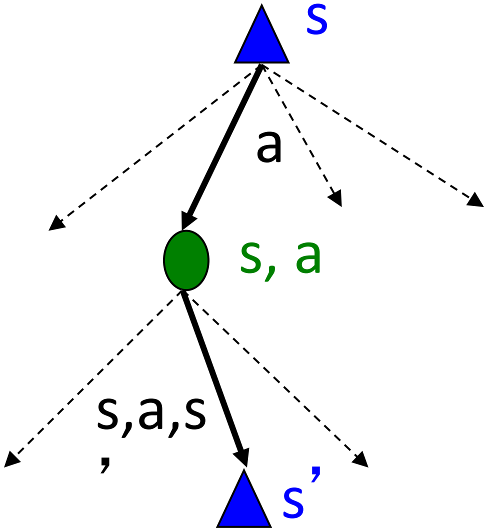
Racing Search Tree (depth=?)



... potentially infinite depth …

Recap: Defining MDPs

- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

Next: Solving MDPs