Exam 3 Review

- Thursday 3:30-5:30pm in class!
- Bring a calculator. You can check out one from the library.
- One page of notes front and back.
Look where we’ve been!

Informed Search: A-Star

Adversarial Search: Alpha-Beta Pruning

Expectimax Search

White to move
Black winning

max

min

10 10 9 100
Look where we’ve been!

MDPs:
Value Iteration, Policy Iteration

Reinforcement Learning:
Q-Learning, Policy Gradients

Environment

Agent

Actions: a

State: s

Reward: r

DQN
AlphaGo
Look where we’ve been!

Bayes’ Nets

- Burglary
- Earthquake

D-Separation

- Alarm

Variable Elimination

John calls

Mary calls

Sampling
Look where we’ve been!

Markov Models

Value of Perfect Information

Hidden Markov Models: Particle Filters

- Umbrella
- Weather
- Forecast = bad
Look where we’ve been!

Behavioral Cloning

Inverse RL

RL from Human Feedback

DAgger
Probability Recap

- Conditional probability
  \[ P(x|y) = \frac{P(x, y)}{P(y)} \]

- Product rule
  \[ P(x, y) = P(x|y)P(y) \]

- Chain rule
  \[ P(X_1, X_2, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\ldots = \prod_{i=1}^{n} P(X_i|X_1, \ldots, X_{i-1}) \]

- \(X, Y\) independent if and only if: \(\forall x, y : P(x, y) = P(x)P(y)\) \(\Rightarrow P(x|y) = P(x)\)

- \(X\) and \(Y\) are conditionally independent given \(Z\) if and only if: \(\forall x, y, z : P(x, y|z) = P(x|z)P(y|z)\)

\(\bowtie\)
Bayes’ Net Semantics

- A directed, acyclic graph, one node per random variable
- A conditional probability table (CPT) for each node
  - A collection of distributions over $X$, one for each combination of parents’ values
    \[ P(X|a_1 \ldots a_n) \]
- Bayes’ nets implicitly encode joint distributions
  - As a product of local conditional distributions
  - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:
    \[
    P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i|\text{parents}(X_i))
    \]
D-separation: Outline
Active / Inactive Paths

### Question: Are X and Y conditionally independent given evidence variables \{Z\}?
- Yes, if X and Y “\(d\)-separated” by Z
- Consider all (undirected) paths from X to Y
- No active paths = independence!

### A path is active if each triple is active:
- Causal chain \(A \rightarrow B \rightarrow C\) where B is unobserved (either direction)
- Common cause \(A \leftarrow B \rightarrow C\) where B is unobserved
- Common effect (aka v-structure)
  \(A \rightarrow B \leftarrow C\) where B or one of its descendents is observed

### All it takes to block a path is a single inactive segment
D-Separation

- Query: \( X_i \perp\!\!\!\perp X_j \mid \{X_{k_1}, \ldots, X_{k_n}\} \) ?

- Check all (undirected!) paths between \( X_i \) and \( X_j \)
  - If one or more active, then independence not guaranteed
  \[ X_i \perp\!\!\!\perp X_j \mid \{X_{k_1}, \ldots, X_{k_n}\} \]
  - Otherwise (i.e. if all paths are inactive), then independence is guaranteed
  \[ X_i \perp\!\!\!\perp X_j \mid \{X_{k_1}, \ldots, X_{k_n}\} \]
Inference by Enumeration vs. Variable Elimination

\[ P(A_1+b) \propto P(A_1+b) \]
\[ = \sum_{C} \sum_{D} P(A_1+b, C, D) \]
\[ = \sum_{C} \sum_{D} P(A)P(C)P(+b|A, C)P(D|+b, C) \]
\[ \sum_{D} = \text{size} \]
\[ = O(2^3) = \text{size} \]

\[ A \]
\[ +B \]
\[ +C \]
\[ +B \]
\[ +C \]
\[ +B \]
\[ -C \]
\[ +B \]
\[ -C \]
\[ +B \]
\[ -C \]
\[ o(2^2) \]
\[ o(2^2) \]
Inference by Enumeration vs. Variable Elimination

▪ Why is inference by enumeration so slow?
  ▪ You join up the whole joint distribution before you sum out the hidden variables

▪ Idea: interleave joining and marginalizing!
  ▪ Called “Variable Elimination”
  ▪ Still NP-hard, but usually much faster than inference by enumeration
Operation 1: Join Factors

- First basic operation: *joining factors*
- Combining factors:
  - Just like a database join
  - Get all factors over the joining variable
  - Build a new factor over the union of the variables involved

- Example: Join on R

\[
P(R) \times P(T|R) \Rightarrow P(R, T)
\]

- Computation for each entry: pointwise products
  \[
  \forall r, t : P(r, t) = P(r) \cdot P(t|r)
  \]
Operation 2: Eliminate

- Second basic operation: marginalization
- Take a factor and sum out a variable
  - Shrinks a factor to a smaller one
  - A projection operation
- Example:

<table>
<thead>
<tr>
<th></th>
<th>+r</th>
<th>+t</th>
<th>0.08</th>
</tr>
</thead>
<tbody>
<tr>
<td>+r</td>
<td>-t</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>-r</td>
<td>+t</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>-r</td>
<td>-t</td>
<td>0.81</td>
<td></td>
</tr>
</tbody>
</table>

\[ P(R, T) \]

\[ \text{sum } R \]

\[ P(T) \]

<table>
<thead>
<tr>
<th></th>
<th>+t</th>
<th>0.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t</td>
<td>0.83</td>
<td></td>
</tr>
</tbody>
</table>
General Variable Elimination

- Query: \[ P(Q|E_1 = e_1, \ldots, E_k = e_k) \]

- Start with initial factors:
  - Local CPTs (but instantiated by evidence)

- While there are still hidden variables (not Q or evidence):
  - Pick a hidden variable H
  - Join all factors mentioning H
  - Eliminate (sum out) H

- Join all remaining factors and normalize
Example

\[ P(B|j,m) \propto P(B,j,m) \]

\[
\begin{array}{cccccc}
P(B) & P(E) & P(A|B,E) & P(j|A) & P(m|A) \\
\end{array}
\]

Choose A

\[ f_1(j,m,B,E) \]

\[
\begin{array}{ccccccc}
P(A|B,E) & P(j,m,A|B,E) & \sum & P(j,m|B,E) \\
\end{array}
\]

\[
\begin{array}{cccc}
P(B) & P(E) & P(j,m|B,E) \\
\end{array}
\]
Example

Choose E

\[
\begin{align*}
P(E) & \quad \times \quad P(j, m, E|B) \quad \sum \quad P(j, m|B)
\end{align*}
\]

Finish with B

\[
\begin{align*}
P(B) & \quad \times \quad P(j, m, B) \quad \text{Normalize} \quad P(B|j, m)
\end{align*}
\]
### Same Example in Equations

**Query**

\[ P(B \mid j, m) \propto P(B, j, m) \]

<table>
<thead>
<tr>
<th>( P(B) )</th>
<th>( P(E) )</th>
<th>( P(A \mid B, E) )</th>
<th>( P(j \mid A) )</th>
<th>( P(m \mid A) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(B, j, m) \propto P(B, j, m) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= ( \sum_{e,a} P(B, j, m, e, a) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= ( \sum_{e,a} P(B)P(e)P(a \mid B, e)P(j \mid a)P(m \mid a) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= ( \sum_{e} P(B)P(e) \sum_{a} P(a \mid B, e)P(j \mid a)P(m \mid a) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= ( \sum_{e} P(B)P(e)f_1(B, e, j, m) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= ( P(B) \sum_{e} P(e)f_1(B, e, j, m) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= ( P(B)f_2(B, j, m) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Marginal can be obtained from joint by summing out:

- by summing out
- use Bayes' net joint distribution expression
- use \( x*(y+z) = xy + xz \)
- joining on \( a \), and then summing out gives \( f_1 \)
- use \( x*(y+z) = xy + xz \)
- joining on \( e \), and then summing out gives \( f_2 \)

All we are doing is exploiting \( uwy + uwz + uxy + uxz + vwy + vwz + vxy + vxz = (u+v)(w+x)(y+z) \) to improve computational efficiency!
Bayes’ Nets: Sampling
**Sampling**

- **Sampling from given distribution**
  - Step 1: Get sample \( u \) from uniform distribution over \([0, 1)\)
  ```python
  >>> import random
  >>> random.random()
  0.6303136415860905
  ```
  - Step 2: Convert this sample \( u \) into an outcome for the given distribution by having each outcome associated with a sub-interval of \([0,1)\) with sub-interval size equal to probability of the outcome

- **Example**

<table>
<thead>
<tr>
<th>( C )</th>
<th>( P(C) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>0.6</td>
</tr>
<tr>
<td>green</td>
<td>0.1</td>
</tr>
<tr>
<td>blue</td>
<td>0.3</td>
</tr>
</tbody>
</table>

  - If random() returns \( u = 0.83 \), then our sample is \( C = \text{blue} \)
  - E.g., after sampling 8 times:
Bayes’ Net Sampling Summary

- Prior Sampling  $P$
- Likelihood Weighting  $P( Q | e)$
- Rejection Sampling  $P( Q | e)$
- Gibbs Sampling  $P( Q | e)$
Prior Sampling

- For $i=1, 2, ..., n$
  - Sample $x_i$ from $P(X_i \mid \text{Parents}(X_i))$
  - Return $(x_1, x_2, ..., x_n)$
Rejection Sampling

- **IN: evidence instantiation**
- **For** $i=1, 2, ..., n$
  - Sample $x_i$ from $P(X_i \mid \text{Parents}(X_i))$
  - If $x_i$ not consistent with evidence
    - Reject: Return, and no sample is generated in this cycle
- **Return** $(x_1, x_2, ..., x_n)$
**Likelihood Weighting**

- **IN:** evidence instantiation
- \( w = 1.0 \)
- for \( i = 1, 2, \ldots, n \)
  - if \( X_i \) is an evidence variable
    - \( X_i = \) observation \( x_i \) for \( X_i \)
    - Set \( w = w \cdot P(x_i | \text{Parents}(X_i)) \)
  - else
    - Sample \( x_i \) from \( P(X_i | \text{Parents}(X_i)) \)
- return \((x_1, x_2, \ldots, x_n), w\)
Likelihood Weighting

- **IN:** evidence instantiation
- w = 1.0
- for i=1, 2, ..., n
  - if \( X_i \) is an evidence variable
    - \( X_i = \) observation \( x_i \) for \( X_i \)
    - Set \( w = w * P(x_i \mid \text{Parents}(X_i)) \)
  - else
    - Sample \( x_i \) from \( P(X_i \mid \text{Parents}(X_i)) \)
- return \((x_1, x_2, ..., x_n), w\)

Now each sample doesn’t count as 1.0 but has a weight. Need to take a weighted average.

\[
P(Q \mid \text{Evidence}) = \frac{\text{Sum(weights of samples consistent with Query)}}{\text{Total Weight of All samples}}
\]
Decision Networks

- **Action selection**
  - Instantiate all evidence
  - Set action node(s) each possible way
  - Calculate posterior for all parents of utility node, given the evidence
  - Calculate expected utility for each action
  - Choose maximizing action
Decision Networks

Umbrella = leave

$$\text{EU(leave)} = \sum_w P(w)U(\text{leave}, w)$$

$$= 0.7 \cdot 100 + 0.3 \cdot 0 = 70$$

Umbrella = take

$$\text{EU(take)} = \sum_w P(w)U(\text{take}, w)$$

$$= 0.7 \cdot 20 + 0.3 \cdot 70 = 35$$

Optimal decision = leave

$$\text{MEU}(\phi) = \max_a \text{EU}(a) = 70$$

<table>
<thead>
<tr>
<th>W</th>
<th>P(W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.7</td>
</tr>
<tr>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>W</th>
<th>U(A,W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>leave</td>
<td>sun</td>
<td>100</td>
</tr>
<tr>
<td>leave</td>
<td>rain</td>
<td>0</td>
</tr>
<tr>
<td>take</td>
<td>sun</td>
<td>20</td>
</tr>
<tr>
<td>take</td>
<td>rain</td>
<td>70</td>
</tr>
</tbody>
</table>
Example: Decision Networks

Umbrella = leave

\[ \text{EU(leave|bad)} = \sum_{w} P(w|\text{bad})U(\text{leave}, w) \]

\[ = 0.34 \cdot 100 + 0.66 \cdot 0 = 34 \]

Umbrella = take

\[ \text{EU(take|bad)} = \sum_{w} P(w|\text{bad})U(\text{take}, w) \]

\[ = 0.34 \cdot 20 + 0.66 \cdot 70 = 53 \]

Optimal decision = take

\[ \text{MEU}(F = \text{bad}) = \max_{a} \text{EU}(a|\text{bad}) = 53 \]
VPI Example: Weather

MEU with no evidence

\[ \text{MEU}(\emptyset) = \max_a \text{EU}(a) = 70 \]

MEU if forecast is bad

\[ \text{MEU}(F = \text{bad}) = \max_a \text{EU}(a|\text{bad}) = 53 \]

MEU if forecast is good

\[ \text{MEU}(F = \text{good}) = \max_a \text{EU}(a|\text{good}) = 95 \]

Forecast distribution

<table>
<thead>
<tr>
<th>F</th>
<th>P(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>0.59</td>
</tr>
<tr>
<td>bad</td>
<td>0.41</td>
</tr>
</tbody>
</table>

\[
\text{VPI}(E'|e) = \left( \sum_{e'} P(e'|e) \text{MEU}(e, e') \right) - \text{MEU}(e)
\]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>W</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>leave</td>
<td>sun</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>leave</td>
<td>rain</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>take</td>
<td>sun</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>take</td>
<td>rain</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>
Assume we have evidence $E=e$. Value if we act now:

$$\text{MEU}(e) = \max_a \sum_s P(s|e) \ U(s, a)$$

Assume we see that $E' = e'$. Value if we act then:

$$\text{MEU}(e, e') = \max_a \sum_s P(s|e, e') \ U(s, a)$$

BUT $E'$ is a random variable whose value is unknown, so we don’t know what $e'$ will be.

Expected value if $E'$ is revealed and then we act:

$$\text{MEU}(e, E') = \sum_{e'} P(e'|e) \text{MEU}(e, e')$$

Value of information: how much MEU goes up by revealing $E'$ first then acting, over acting now:

$$\text{VPI}(E'|e) = \text{MEU}(e, E') - \text{MEU}(e)$$
Markov Models Recap

- Explicit assumption for all $t$: $X_t \perp X_1, \ldots, X_{t-2} \mid X_{t-1}$

- Consequence, joint distribution can be written as:

$$P(X_1, X_2, \ldots, X_T) = P(X_1)P(X_2|X_1)P(X_3|X_2)\ldots P(X_T|X_{T-1})$$

$$= P(X_1) \prod_{t=2}^{T} P(X_t|X_{t-1})$$

Huge savings in number of parameters needed!

- Implied conditional independencies:
  - Past variables independent of future variables given the present
  i.e., if $t_1 < t_2 < t_3$ or $t_1 > t_2 > t_3$ then: $X_{t_1} \perp X_{t_3} \mid X_{t_2}$

- Additional explicit assumption: $P(X_t \mid X_{t-1})$ is the same for all $t$
Mini-Forward Algorithm

- Question: What’s $P(X)$ on some day $t$?

$$P(x_1) = \text{known}$$

$$P(x_t) = \sum_{x_{t-1}} P(x_{t-1}, x_t)$$

$$= \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1})$$

Forward simulation
Stationary Distributions

- **For most chains:**
  - Influence of the initial distribution gets less and less over time.
  - The distribution we end up in is independent of the initial distribution.

- **Stationary distribution:**
  - The distribution we end up with is called the *stationary distribution* $P_\infty$ of the chain.
  - It satisfies

\[
P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)
\]

\[
\sum_x P_\infty(x) = 1
\]
HMMs Recap

- Explicit assumption for all $t$: $X_t \perp X_1, \ldots, X_{t-2} \mid X_{t-1}$

- Consequence, joint distribution can be written as:

$$P(X_1, X_2, \ldots, X_T) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_2)\cdots P(X_T \mid X_{T-1})$$

$$= P(X_1)\prod_{t=2}^{T} P(X_t \mid X_{t-1})$$

- Implied conditional independencies:
  - Past variables independent of future variables given the present
  i.e., if $t_1 < t_2 < t_3$ or $t_1 > t_2 > t_3$ then: $X_{t_1} \perp X_{t_3} \mid X_{t_2}$

- Additional explicit assumption: $P(X_t \mid X_{t-1})$ is the same for all $t$
The Forward Algorithm

- We are given evidence at each time and want to know

\[ B_t(X) = P(X_t|e_{1:t}) \]

- We can derive the following recursive update

\[
P(x_t|e_{1:t}) = P(x_t|e_{1:t-1}, e_t) \]

\[ \propto P(e_t|x_t, e_{1:t-1})P(x_t|e_{1:t-1}) \]

\[ = P(e_t|x_t)P(x_t|e_{1:t-1}) \]

\[ = P(e_t|x_t) \sum_{x_{t-1}} P(x_t, x_{t-1}|e_{1:t-1}) \]

\[ = P(e_t|x_t) \sum_{x_{t-1}} P(x_t|e_{1:t-1}, x_{t-1})P(x_{t-1}|e_{1:t-1}) \]

\[ = P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1})P(x_{t-1}|e_{1:t-1}) \]

Divide up evidence
Bayes’ rule
Sensor Markov assumption
Reverse marginalization
Product rule
Markov assumption
Particle Filtering
Particle Filtering

- Filtering: approximate solution
- Sometimes $|X|$ is too big to use exact inference
  - $|X|$ may be too big to even store $B(X)$
  - E.g. $X$ is continuous
- Solution: approximate inference
  - Track samples of $X$, not all values
  - Samples are called particles
  - Time per step is linear in the number of samples
  - But: number needed may be large
  - In memory: list of particles, not states
- This is how robot localization works in practice
- Particle is just new name for sample
Our representation of $P(X)$ is now a list of $N$ particles (samples)
- Generally, $N << |X|$
- Storing map from $X$ to counts would defeat the point

$P(x)$ approximated by number of particles with value $x$
- So, many $x$ may have $P(x) = 0!$
- More particles, more accuracy

For now, all particles have a weight of 1
Particle Filtering: Elapse Time

- Each particle is moved by sampling its next position from the transition model

\[ x' = \text{sample}(P(X'|x)) \]

- This is like prior sampling – samples’ frequencies reflect the transition probabilities

- Here, most samples move clockwise, but some move in another direction or stay in place

- This captures the passage of time
  - If enough samples, close to exact values before and after (consistent)
Particle Filtering: Observe

- Slightly trickier:
  - Don’t sample observation, fix it
  - Similar to likelihood weighting, downweight samples based on the evidence

\[
 w(x) = P(e|x) 
\]

\[
 B(X) \propto P(e|X)B'(X) 
\]

- As before, the probabilities don’t sum to one, since all have been downweighted (in fact they now sum to (N times) an approximation of P(e))
Rather than tracking weighted samples, we resample.

N times, we choose from our weighted sample distribution (i.e. draw with replacement).

This is equivalent to renormalizing the distribution.

Now the update is complete for this time step, continue with the next one.
Recap: Particle Filtering

- **Particles**: track samples of states rather than an explicit distribution

Elapse $P(x_t | x_{t-1})$ Weight $P(e_t | x_t)$ Resample

Particles:
- (3,3)
- (2,3)
- (3,3)
- (3,2)
- (3,3)
- (3,3)
- (3,1)
- (3,3)
- (3,2)
- (1,2)
- (2,2)

(Demos: ghostbusters particle filtering (L15D3,4,5))
Partially Observable Markov Decision Processes

A POMDP is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- 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
- A reward function $R(s, a, s')$
  - Sometimes just $R(s)$, $R(s,a)$, or $R(s')$
- A start state distribution
- Maybe a terminal state
- Observations $Z$
- Emission Model $O(s,z) = P(z | s)$

POMDPs are non-deterministic search problems where you don’t know where you are!
MDP vs POMDP

- **MDP**
  - + Tractable to solve
  - + Relatively easy to specify
  - - Assumes perfect knowledge of state

- **POMDP**
  - + Models the real world
  - + Allows for information gathering actions
  - - Hugely intractable to solve optimally
Belief State MDP

- **State space:** \( B \)
- **Action space:** \( A \)
- **Transition Function:** \( P(b'|b,a) \)

\[
P(b'|b,a) = \sum_z P(b',z|b,a) = \sum_z P(b'|b,a,z)P(z|b,a)
\]

0 or 1 depending on state estimation

- **Reward function:**

\[
R(b,a) = \sum_s b(s)r(s,a)
\]

- **Problems?**
Behavioral Cloning

\[ \pi_\theta(a_t | o_t) \]

- \( o_t \)
- \( a_t \)

training data

supervised learning

- Training trajectory
- \( \pi_\theta \) expected trajectory
Distribution Shift

\[ p_{\pi^*}(o_t) \neq p_{\pi_\theta}(o_t) \]

<table>
<thead>
<tr>
<th></th>
<th>Supervised Learning</th>
<th>Supervised Learning + Control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Train</strong></td>
<td>((x, y) \sim D)</td>
<td>(s \sim P(\cdot</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td>((x, y) \sim D)</td>
<td>(s \sim P(\cdot</td>
</tr>
</tbody>
</table>
can we make $p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t)$?
idea: instead of being clever about $p_{\pi_\theta}(o_t)$, be clever about $p_{\text{data}}(o_t)$!

**DAgger: Dataset Aggregation**

goal: collect training data from $p_{\pi_\theta}(o_t)$ instead of $p_{\text{data}}(o_t)$
how? just run $\pi_\theta(a_t|o_t)$
but need labels $a_t$!

1. train $\pi_\theta(a_t|o_t)$ from human data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, \ldots, o_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $a_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Ross et al. ‘11
Behavioral Cloning

- Answers the “How?” question
- Mimic the demonstrator
- Learn mapping from states to actions
- Computationally efficient
- Compounding errors

\[ \Rightarrow \pi \]

Inverse Reinforcement Learning

- Answers the “Why?” question
- Explain the demonstrator’s behavior
- Learn a reward function capturing the demonstrator’s intent
- Can require lots of data and compute
- Better generalization. Can recover from arbitrary states

\[ \Rightarrow R \Rightarrow \pi \]
Basic IRL Algorithm

- Start with demonstrations, $D$
- Guess initial reward function $R_0$
- $\hat{R} = R_0$
- Loop:
  - Solve for optimal policy $\pi^*_\hat{R}$
  - Compare $D$ and $\pi^*_\hat{R}$
  - Update $\hat{R}$ to try and make $D$ and $\pi^*_\hat{R}$ more similar
RL from Human Feedback (RLHF)
RL from Human Preferences
Pre-ranked demonstrations

Pre-ranked demonstrations

T-REX Policy

Learning from preferences

\[ \tau_1 \prec \tau_2 \prec \cdots \prec \tau_T \]

Bradley-Terry pairwise ranking loss

\[
\mathcal{L}(\theta) = - \sum_{\tau_i < \tau_j} \exp \sum_{s \in \tau_i} R_\theta(s) + \exp \sum_{s \in \tau_j} R_\theta(s)
\]

\[
\sum_{s \in \tau_1} R_\theta(s) < \sum_{s \in \tau_2} R_\theta(s)
\]
Step 1: Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2: Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3: Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.
We made it!