Reinforcement Learning & Apprenticeship Learning

Chenyi Chen
Markov Decision Process (MDP)

- **What’s MDP?**
- A sequential decision problem
- Fully observable, stochastic environment
- Markovian transition model: the $n^{\text{th}}$ state is only determined by $(n-1)^{\text{th}}$ state and $(n-1)^{\text{th}}$ action
- Each state has a reward, and the reward is additive
Markov Decision Process (MDP)

- **State** $s$: a representation of current environment;
Markov Decision Process (MDP)

• Example: Tom and Jerry, control Jerry (Jerry’s perspective)
• State: the position of Tom and Jerry, 25*25=625 in total;
Markov Decision Process (MDP)

- **State** \( s \): a representation of current environment;
- **Action** \( a \): the action can be taken by the agent in state \( s \);
Markov Decision Process (MDP)

- **Example:** Tom and Jerry, control Jerry (Jerry’s perspective)
- **State:** the position of Tom and Jerry, 25*25=625 in total;
- **Action:** both can move to the neighboring 8 squares or stay;
Markov Decision Process (MDP)

- **State** \( s \): a representation of current environment;
- **Action** \( a \): the action can be taken by the agent in state \( s \);
- **Reward** \( R(s) \): the reward of current state \( s \) (+, -, 0);
- **Value (aka utility) of state** \( s \): different from reward, related with future optimal actions;
An Straightforward Example

- 100 bucks if you came to class
- Reward of “come to class” is 100
- You can use the money to:
  - Eat food (you only have 50 bucks left)
  - Stock market (you earn 1000 bucks, including the invested 100 bucks)
- The value (utility) of “come to class” is 1000
Markov Decision Process (MDP)

- **Example**: Tom and Jerry, control Jerry (Jerry’s perspective)
- **State**: the position of Tom and Jerry, $25 \times 25 = 625$ in total;
- **Action**: both can move to the neighboring 8 squares or stay;
- **Reward**: 1) Jerry and cheese at the same square, +5;
  2) Tom and Jerry at the same square, -20;
  3) otherwise 0;

One of the states

![One of the states](image1)

![One of the states](image2)
Markov Decision Process (MDP)

- **State** $s$: a representation of current environment;
- **Action** $a$: the action can be taken by the agent in state $s$;
- **Reward** $R(s)$: the reward of current state $s$ ($+,-,0$);
- **Value (aka utility) of state** $s$: different from reward, related with future optimal actions;
- **Transition probability** $P(s'|s,a)$: given the agent is in state $s$ and taking action $a$, the probability of reaching state $s'$ in the next step;
Markov Decision Process (MDP)

- Example: Tom and Jerry, control Jerry (Jerry’s perspective)
- State: the position of Tom and Jerry, 25*25=625 in total;
- Action: both can move to the neighboring 8 squares or stay;
- Reward: 1) Jerry and cheese at the same square, +5;
  2) Tom and Jerry at the same square, -20;
  3) otherwise 0;
- Transition probability: about Tom’s moving pattern.
Markov Decision Process (MDP)

- Example: Tom and Jerry, control Jerry (Jerry’s perspective)
Markov Decision Process (MDP)

- **State $s$:** a representation of current environment;
- **Action $a$:** the action can be taken by the agent in state $s$;
- **Reward $R(s)$:** the reward of current state $s$ (+,-,0);
- **Value (aka utility) of state $s$:** different from reward, related with future optimal actions;
- **Transition probability $P(s'|s,a)$:** given the agent is in state $s$ and taking action $a$, the probability of reaching state $s'$ in the next step;
- **Policy $\pi(s)\rightarrow a$:** a table of state-action pairs, given state $s$, output action $a$ that should be taken.
**Bellman Equation**

- The expected utility of state $s$ obtained by executing $\pi$ starting in $s$ is given by ($\gamma$ is a discount factor):
  \[
  U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t) \right], \text{ where } S_0 = s
  \]
- The optimal policy is given by:
  \[
  \pi^*_s = \arg\max_{\pi} U^\pi(s)
  \]
- Denote $U^{\pi^*}(s)$ as $U(s)$, the optimal policy chooses the action that maximizes the expected utility of the subsequent state:
  \[
  \pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')
  \]
Bellman Equation

• Bellman Equation:

\[ U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \]

• The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent choose the optimal action.

• \( U^{\pi^*}(s) = E\left[ \sum_{t=0}^{\infty} \gamma^t R(S_t) \right] \) with \( S_0 = s \), is the unique solution to Bellman equation.
Value Iteration

initialize \( U' = 0 \), \( \gamma \) as a discount factor

repeat

\[
U \leftarrow U'; \quad \delta \leftarrow 0
\]

for each state \( s \) in \( S \) do

\[
U'[s] \leftarrow R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s'|s, a)U[s']
\]

\[
\pi[s] \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|s, a)U[s']
\]

if \( |U'[s] - U[s]| > \delta \) then \( \delta \leftarrow |U'[s] - U[s]| \)

until \( \delta < \epsilon (1 - \gamma) / \gamma \)

return \( U, \pi \)

Bellman Equation:

\[
U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s'|s, a)U(s')
\]
Value Iteration

- **Naïve example:** \( R(s)=3, R(s')=5, \gamma=0.9 \)
  - Initially \( U(s)=0, U(s')=0 \)
  - (1) \( U(s)=3+0.9*0=3, U(s')=5+0.9*3=7.7 \)
  - (2) \( U(s)=3+0.9*7.7=9.93, U(s')=5+0.9*9.93=13.937 \)
  - (3) \( U(s)=3+0.9*13.937=15.5433, U(s')=5+0.9*15.5433=18.989 \)
  - ...
  - (29) \( U(s)=39.3738, U(s')=40.4364 \)
  - (30) \( U(s)=39.3928, U(s')=40.4535 \)

Value iteration

\[
U'[s] \leftarrow R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s'|s, a)U[s']
\]

- Solve the equation:
  \[
  \begin{cases}
  U(s)=3+0.9\cdot U(s') \\
  U(s')=5+0.9\cdot U(s)
  \end{cases}
  \]
  - the true value is:
    \( U(s)=39.4737, U(s')=40.5263 \)
Reinforcement Learning

• Similar to MDPs
• But we assume the environment model (transition probability $P(s'|s,a)$) is unknown
Reinforcement Learning

• How to solve it?

• Solution #1: Use Monte Carlo method to sample the transition probability, then implement Value Iteration

  limitation: too slow for problems with many possible states because it ignores frequencies of states
Monte Carlo Method

- A broad class of computational algorithms that rely on repeated random sampling to obtain numerical results;
- Typically one runs simulations many times in order to obtain the distribution of an unknown probabilistic entity.

From Wikipedia
Monte Carlo Example

\[ P(s, a, s') : \text{the element is the probability } P(s'|s,a) \]

**initialize** table with all elements \( P(s, a, s') = \varepsilon, \varepsilon > 0 \)

**repeat**

- at current state \( s \), random choose a valid action \( a \)
- simulate for one step, get a new state \( s' \)
\[ P(s, a, s') \leftarrow P(s, a, s') + 1 \]
\[ s \leftarrow s' \]

**until** sampled enough times

\[ P(s, a, s') \leftarrow P(s, a, s') / \sum_{s'} P(s, a, s') \]

**return** \( P(s, a, s') \)
Reinforcement Learning

• How to solve it?
• Solution #1: Use Monte Carlo method to sample the transition probability, then implement Value Iteration
  limitation: too slow for problems with many possible states because it ignores frequencies of states
• Solution #2: Q-learning
  the major algorithm for reinforcement learning
**Q-learning**

Bellman Equation:

\[
U(s) = R(s) + \gamma \cdot \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')
\]

- **Q-value** is defined by:

\[
Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')
\]

- The relationship between utility and Q-value is:

\[
U(s) = \max_a Q(s,a)
\]

- The optimal policy is given by:

\[
\pi^*(s) = \arg \max_a Q(s,a)
\]

- **Q-learning algorithm** is used to learn this Q-value table
**Q-learning**

$Q$: a table of $Q$-values indexed by state and action, initially zero

$s, a, R(s)$: state, action, and reward. Initial state is given by the environment, and initial action is randomly picked up

$\gamma$: discount factor

$\alpha$: learning rate

$f(.)$: greedy function, at the beginning, $Q$-table is bad, so we make some random choice

**While not coverage**

run one step to obtain $s'$ from $s$ and $a$ through the environment (e.g. the game engine)

$$Q[s, a] \leftarrow Q[s, a] + \alpha \cdot ((R(s) + \gamma \cdot \max_{a'} Q[s', a']) - Q[s, a])$$

$s, a, R(s) \leftarrow s', f(\arg\max_{a'} Q[s', a']), R(s')$

**return** $Q$

$Q$-value is defined by:

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a)\max_{a'} Q(s', a')$$
Playing **Atari** with Deep Reinforcement Learning

- The Atari 2600 is a video game console released in September 1977 by Atari, Inc.
- Atari emulator: Arcade Learning Environment (ALE)
What did they do?

• Train a deep learning convolutional neural network
• Input is current state (raw image sequence)
• Output is all the legal action and corresponding $Q(s,a)$ value
• Let the CNN play Atari games
What’s Special?

• Input is raw image!
• Output is the action!
• Game independent, same convolutional neural network for all games
• Outperform human expert players in some games
Problem Definition

- State: $S_t = x_1, a_1, x_2, ..., x_{t-1}, a_{t-1}, x_t$
- Action: possible actions in the game
- Reward: score won in the Atari games (output of the emulator)
- Learn the optimal policy through training
A Variant of $Q$-learning

In the paper:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right]$$

$Q$-value is defined by:

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$$
Deep Learning Approach

Approach the $Q$-value with a convolutional neural network $Q(s,a;\theta)$

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Straightforward structure

The structure used in the paper
How to Train the Convolutional Neural Network?

Loss function:

\[ L_i (\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right] \]

Where:

\[ y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_i) \middle| s, a \right] \]

Q-value is defined as:

\[ Q^*(s, a) = \mathbb{E}_{s', a' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right] \]

\[ \nabla_{\theta_i} L_i (\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta_i) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \]

Do gradient descent:

\[ \theta_{i+1} = \theta_i + \alpha \cdot \nabla_{\theta_i} L_i (\theta_i) \]
Some Details

- The distribution of action $a$ ($\epsilon$-greedy policy): choose a “best” action with probability $1-\epsilon$, and selects a random action with probability $\epsilon$, $\epsilon$ annealed linearly from 1 to 0.1
- Input image preprocessing function $\phi(s_t)$
- Build a huge database to store historical samples

Database $D$ of samples

(\(\phi_{k1}, a_{k1}, r_{k1}, \phi_{k1+1}\))  
(\(\phi_{k2}, a_{k2}, r_{k2}, \phi_{k2+1}\))  
\(\cdots\)  
(\(\phi_{kn}, a_{kn}, r_{kn}, \phi_{kn+1}\))

1 million samples
During Training...

Database $D$ of samples
$(\phi_s, a_s, r_s, \phi_{s+1})$
1 million samples

Do mini-batch gradient descent on parameter $\theta$
for one step

Input game image

Under training Convolutional Neural Network Parameter $\theta$

Play the game for one step

Database $D$ of samples
$(\phi_k, a_k, r_k, \phi_{k+1})$
$n$=mini-batch size

$Q(s_t, a_{tl})$ & $a_{tl}$

$Q(s_t, a_{l2})$ & $a_{l2}$

$Q(s_t, a_{lm})$ & $a_{lm}$

Add new data sample to database

$(\phi_{t-1}, a_{t-1}, r_{t-1}, \phi_t)$

$a_t^* = \arg\max_a Q(s_t, a)$
with probability $1-\varepsilon$

or
random action $a_t$
with probability $\varepsilon$
CNN Training Pipeline

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights

for episode = 1, $M$ do
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ do
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for
After Training...

Train a Convolutional Neural Network with parameters $\theta$

Input game image

Play the game

$Q(s,a_{s1}) & a_{s1}$

$Q(s,a_{s2}) & a_{s2}$

$Q(s,a_{sn}) & a_{sn}$

$a^*(s) = \arg\max_a Q(s,a)$
Results

Screen shots from five Atari 2600 games: (Left-to-right) Beam Rider, Breakout, Pong, Seaquest, Space Invaders

Comparison of average total reward for various learning methods by running an $\epsilon$-greedy policy with $\epsilon = 0.05$ for a fixed number of steps

<table>
<thead>
<tr>
<th>Method</th>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>354</td>
<td>1.2</td>
<td>0</td>
<td>−20.4</td>
<td>157</td>
<td>110</td>
<td>179</td>
</tr>
<tr>
<td>Sarsa</td>
<td>996</td>
<td>5.2</td>
<td>129</td>
<td>−19</td>
<td>614</td>
<td>665</td>
<td>271</td>
</tr>
<tr>
<td>Contingency</td>
<td>1743</td>
<td>6</td>
<td>159</td>
<td>−17</td>
<td>960</td>
<td>723</td>
<td>268</td>
</tr>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>Human</td>
<td>7456</td>
<td>31</td>
<td>368</td>
<td>−3</td>
<td>18900</td>
<td>28010</td>
<td>3690</td>
</tr>
</tbody>
</table>
Results

• The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.
Apprenticeship Learning via Inverse Reinforcement Learning

• Teach the computer to do something by demonstration, rather than by telling it the rules or reward

• **Reinforcement Learning:** tell computer the reward, let it learn by itself using the reward

• **Apprenticeship Learning:** demonstrate to the computer, let it mimic the performance
Why Apprenticeship Learning?

• For standard MDPs, a reward for each state needs to be specified
• Specify a reward some time is not easy, what’s the reward for driving?
• When teaching people to do something (e.g. driving), usually we prefer to demonstrate rather than tell them the reward function
How Does It Work?

- Reward is unknown, but we assume it’s a linear function of features, $\phi : S \rightarrow [0, 1]^k$ is a function mapping state $s$ to features, so:

$$R^*(s) = w^* \cdot \phi(s), \text{ where } w^* \in \mathbb{R}^k$$
Example of Feature

- **State** $S_t$ of the red car is defined as:
  
  $S_t = 1$ left lane, $S_t = 2$ middle lane, $S_t = 3$ right lane

- **Feature** $\phi(s_t)$ is defined as:
  
  $[1 \ 0 \ 0]$ left lane, $[0 \ 1 \ 0]$ middle lane, $[0 \ 0 \ 1]$ right lane

- **$w$** is defined as:
  
  $w = [0.1 \ 0.5 \ 0.3]$

  $R($left lane$)=0.1$, $R($middle lane$)=0.5$, $R($right lane$)=0.3$

- So in this case staying in the middle lane is preferred

$$R^*(s) = w^* \cdot \phi(s), \text{ where } w^* \in \mathbb{R}^k$$
How Does It Work?

• Reward is unknown, but we assume it’s a linear function of features, \( \phi : S \rightarrow [0, 1]^k \) is a function mapping state \( s \) to features, so:

\[
R^*(s) = w^* \cdot \phi(s), \quad \text{where } w^* \in \mathbb{R}^k
\]

• The value (utility) of policy \( \pi \) is:

\[
E_{s_0 \sim D} [V^\pi (s_0)] = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi \right] = E \left[ \sum_{t=0}^{\infty} \gamma^t w \cdot \phi(s_t) \mid \pi \right] = w \cdot E \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi \right]
\]

The expected utility obtained by executing \( \pi \) starting in \( s \) is given by:

\[
U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t) \right], \quad \text{where } S_0 = s
\]
How Does It Work?

• Define **feature expectation** as:

\[ \mu(\pi) = E\left[ \sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi \right] \in \mathbb{R}^k \]

• Then:

\[ E_{s_0 \sim D}[V^\pi(s_0)] = w \cdot \mu(\pi) \]

• Assume the expert’s demonstration defines the optimal policy:

\[ \mu_E = \mu(\pi_E) \]

• We need to sample the expert’s feature expectation by (sample \(m\) times):

\[ \hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \]
What Does Feature Expectation Look Like?

- **State** $S_t$ of the red car is defined as:
  
  $S_t = 1$ left lane, $S_t = 2$ middle lane, $S_t = 3$ right lane

- **Feature** $\phi(s_t)$ is defined as:
  
  $[1 \ 0 \ 0]$ left lane, $[0 \ 1 \ 0]$ middle lane, $[0 \ 0 \ 1]$ right lane

- **During sampling, assume** $\gamma = 0.9$

  **Step 1**, red car in middle lane
  
  $\mu = 0.9^0 [0 \ 1 \ 0] = [0 \ 1 \ 0]$

  **Step 2**, red car still in middle lane
  
  $\mu = [0 \ 1 \ 0] + 0.9^1 [0 \ 1 \ 0] = [0 \ 1.9 \ 0]$

  **Step 3**, red car move to left lane
  
  $\mu = [0 \ 1.9 \ 0] + 0.9^2 [1 \ 0 \ 0] = [0.81 \ 1.9 \ 0]$

  ...

\[
\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)})
\]
How Does It Work?

• We want to mimic the expert’s performance by minimize the difference between

\[ E_{s_0 \sim D}[V^\pi_E(s_0)] \quad \text{and} \quad E_{s_0 \sim D}[V^{\tilde{\pi}}(s_0)] \]

\[ E_{s_0 \sim D}[V^\pi(s_0)] = E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] \]

• If we have \[ \|\mu(\tilde{\pi}) - \mu_E\|_2 \leq \epsilon \], and assuming \[ ||w||_2 \leq 1 \]

Then

\[ |E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi_E] - E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\tilde{\pi}]| \]

\[ = |w^T \mu(\tilde{\pi}) - w^T \mu_E| \]

\[ \leq ||w||_2 \|\mu(\tilde{\pi}) - \mu_E\|_2 \]

\[ \leq 1 \cdot \epsilon = \epsilon \]
Pipeline

\[ E_{s_0 \sim D}[V^{\pi}(s_0)] = w \cdot \mu(\pi) \]

1. Randomly pick some policy \( \pi^{(0)} \), compute (or approximate via Monte Carlo) \( \mu^{(0)} = \mu(\pi^{(0)}) \), and set \( i = 1 \).

2. Compute \( t^{(i)} = \max_{w: \|w\|_2 \leq 1} \min_{j \in \{0 \ldots (i-1)\}} w^T (\mu_E - \mu^{(j)}) \), and let \( w^{(i)} \) be the value of \( w \) that attains this maximum.

3. If \( t^{(i)} \leq \epsilon \), then terminate.

4. Using the RL algorithm, compute the optimal policy \( \pi^{(i)} \) for the MDP using rewards \( R = (w^{(i)})^T \phi \).

5. Compute (or estimate) \( \mu^{(i)} = \mu(\pi^{(i)}) \).

6. Set \( i = i + 1 \), and go back to step 2.

Upon termination, the algorithm returns \( \{\pi^{(i)} : i = 0 \ldots n\} \)
Supporting Vector Machine (SVM)

- The 2\textsuperscript{nd} step of the pipeline is a SVM problem

\[
2. \text{Compute } t^{(i)} = \max_{w : \|w\|_2 \leq 1} \min_{j \in \{0, \ldots, (i-1)\}} \ w^T (\mu_E - \mu^{(j)}) \text{, and let } w^{(i)} \text{ be the value of } w \text{ that attains this maximum.}
\]

\[
E_{s_0} \sim D[V^\pi(s_0)] = w \cdot \mu(\pi)
\]

Which can be rewritten as:

\[
\max_{t, w} \ t \\
\text{s.t. } w^T \mu_E \geq w^T \mu^{(j)} + t, \ j = 0, \ldots, i - 1 \\
\|w\|_2 \leq 1
\]
 Pipeline

Sample expert’s performance $\mu_E$

Random initial policy $\pi(0)$

Sample policy $\pi(i)$’s performance $\mu(i)$

SVM

Get $w(i)$ and $t(i)$

$R = (w(i))^T \phi$

and RL algorithm to produce a new policy $\pi(i)$

Terminate if $t(i) \leq \varepsilon$

$\{\pi(i) : i = 0 \ldots n\}$
Their Testing System
## Demo Videos

http://ai.stanford.edu/~pabbeel/irl/

<table>
<thead>
<tr>
<th>Driving Style</th>
<th>Expert</th>
<th>Learned Controller</th>
<th>Both (Expert left, Learned right)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Nice</td>
<td>expert1.avi</td>
<td>learnedcontroller1.avi</td>
<td>joined1.avi</td>
</tr>
<tr>
<td>2: Nasty</td>
<td>expert2.avi</td>
<td>learnedcontroller2.avi</td>
<td>joined2.avi</td>
</tr>
<tr>
<td>3: Right lane nice</td>
<td>expert3.avi</td>
<td>learnedcontroller3.avi</td>
<td>joined3.avi</td>
</tr>
<tr>
<td>4: Right lane nasty</td>
<td>expert4.avi</td>
<td>learnedcontroller4.avi</td>
<td>joined4.avi</td>
</tr>
<tr>
<td>5: Middle lane</td>
<td>expert5.avi</td>
<td>learnedcontroller5.avi</td>
<td>joined5.avi</td>
</tr>
</tbody>
</table>
Their Results

Expert’s performance $\hat{\mu}_E$, learnt policy’s performance $\mu(\tilde{\pi})$, and feature weight $\tilde{w}$

<table>
<thead>
<tr>
<th>1: Nice</th>
<th>Collision</th>
<th>Offroad Left</th>
<th>LeftLane</th>
<th>MiddleLane</th>
<th>RightLane</th>
<th>Offroad Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\mu}_E$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1325</td>
<td>0.2033</td>
<td>0.5983</td>
<td>0.0658</td>
</tr>
<tr>
<td>$\mu(\tilde{\pi})$</td>
<td>0.0001</td>
<td>0.0004</td>
<td>0.0904</td>
<td>0.2287</td>
<td>0.6041</td>
<td>0.0764</td>
</tr>
<tr>
<td>$\tilde{w}$</td>
<td>-0.0767</td>
<td>-0.0439</td>
<td>0.0077</td>
<td>0.0078</td>
<td>0.0318</td>
<td>-0.0035</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2: Nasty</th>
<th>Collision</th>
<th>Offroad Left</th>
<th>LeftLane</th>
<th>MiddleLane</th>
<th>RightLane</th>
<th>Offroad Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\mu}_E$</td>
<td>0.1167</td>
<td>0.0000</td>
<td>0.0633</td>
<td>0.4667</td>
<td>0.4700</td>
<td>0.0000</td>
</tr>
<tr>
<td>$\mu(\tilde{\pi})$</td>
<td>0.1332</td>
<td>0.0000</td>
<td>0.1045</td>
<td>0.3196</td>
<td>0.5759</td>
<td>0.0000</td>
</tr>
<tr>
<td>$\tilde{w}$</td>
<td>0.2340</td>
<td>-0.1098</td>
<td>0.0092</td>
<td>0.0487</td>
<td>0.0576</td>
<td>-0.0056</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3: Right lane nice</th>
<th>Collision</th>
<th>Offroad Left</th>
<th>LeftLane</th>
<th>MiddleLane</th>
<th>RightLane</th>
<th>Offroad Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\mu}_E$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0033</td>
<td>0.7058</td>
<td>0.2908</td>
</tr>
<tr>
<td>$\mu(\tilde{\pi})$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.7447</td>
<td>0.2554</td>
</tr>
<tr>
<td>$\tilde{w}$</td>
<td>-0.1056</td>
<td>-0.0051</td>
<td>-0.0573</td>
<td>-0.0386</td>
<td>0.0929</td>
<td>0.0081</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4: Right lane nasty</th>
<th>Collision</th>
<th>Offroad Left</th>
<th>LeftLane</th>
<th>MiddleLane</th>
<th>RightLane</th>
<th>Offroad Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\mu}_E$</td>
<td>0.0600</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0033</td>
<td>0.2908</td>
<td>0.7058</td>
</tr>
<tr>
<td>$\mu(\tilde{\pi})$</td>
<td>0.0569</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2666</td>
<td>0.7334</td>
</tr>
<tr>
<td>$\tilde{w}$</td>
<td>0.1079</td>
<td>-0.0001</td>
<td>-0.0487</td>
<td>-0.0666</td>
<td>0.0590</td>
<td>0.0564</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5: Middle lane</th>
<th>Collision</th>
<th>Offroad Left</th>
<th>LeftLane</th>
<th>MiddleLane</th>
<th>RightLane</th>
<th>Offroad Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\mu}_E$</td>
<td>0.0600</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>$\mu(\tilde{\pi})$</td>
<td>0.0542</td>
<td>0.0000</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>$\tilde{w}$</td>
<td>0.0094</td>
<td>-0.0108</td>
<td>-0.2765</td>
<td>0.8126</td>
<td>-0.5099</td>
<td>-0.0154</td>
</tr>
</tbody>
</table>
Questions?