Imitation and Inverse Reinforcement Learning

- Today:

Imitation and Inverse Reinforcement Learning

- Today:
  - What if you don't know the reward function and just want to act like an expert?
    - Imitation Learning
    - Inverse Reinforcement Learning

Trivia: When was the first car driven with a Neural Network?

1997

1992

1967

1902

Trivia: When was the first car driven with a Neural Network?

Dean Pomerleau
@deanpomerleau

Replying to @GTARobotics

GPU? Gez, ALVINN ran on 100 MFLOP CPU, ~10x slower than iWatch; Refrigerator-size & needed 5000 watt generator. @olivercameron

What's Hidden in the Hidden Layers?

The contents can be easy to find with a geometrical problem, but the hidden layers have yet to give up all their secrets

David S. Touretzky and Dean A. Pomerleau

AUGUST 1989 • BYTE 231

tions, we fed the network road images taken under a wide variety of viewing angles and lighting conditions. It would be impractical to try to collect thousands of real road images for such a data set. Instead, we developed a synthetic road-image generator that can create as many training examples as we need.

To train the network, 1200 simulated road images are presented 40 times each, while the weights are adjusted using the back-propagation learning algorithm. This takes about 30 minutes on Carnegie Mellon's Warp systolic-array supercomputer. (This machine was designed at Carnegie Mellon and is built by General Electric. It has a peak rate of 100 million floating-point operations per second and can compute weight adjustments for back-propagation networks at a rate of 20 million connections per second.)

Once it is trained, ALVINN can accurately drive the NAVLAB vehicle at about 3½ miles per hour along a path through a wooded area adjoining the Carnegie Mellon campus, under a variety of weather and lighting conditions. This speed is nearly twice as fast as that achieved by non-neural-network algorithms running on the same vehicle. Part of the reason for this is that the forward pass of a back-propagation network can be computed quickly. It takes about 200 milliseconds on the Sun-3/160 workstation installed on the NAVLAB.

The hidden-layer representations ALVINN develops are interesting. When trained on roads of a fixed width, the network chooses a representation in which hidden units act as detectors for complete roads at various positions and orientations. When trained on roads of variable

continued

Photo 1: *The NAVLAB autonomous navigation test-bed vehicle and the road used for trial runs.*

Trivia: When was the first car driven with a Neural Network?

Dean Pomerleau
@deanpomerleau

Replying to @GTARobotics

GPU? Gez, ALVINN ran on 100 MFLOP CPU, ~10x slower than iWatch; Refrigerator-size & needed 5000 watt generator. @olivercameron

What's Hidden in the Hidden Layers?

The contents can be easy to find with a geometrical problem, but the hidden layers have yet to give up all their secrets

David S. Touretzky and Dean A. Pomerleau

AUGUST 1989 • BYTE 231

tions, we fed the network road images taken under a wide variety of viewing angles and lighting conditions. It would be impractical to try to collect thousands of real road images for such a data set. Instead, we developed a synthetic road-image generator that can create as many training examples as we need.

To train the network, 1200 simulated road images are presented 40 times each, while the weights are adjusted using the back-propagation learning algorithm. This takes about 30 minutes on Carnegie Mellon's Warp systolic-array supercomputer. (This machine was designed at Carnegie Mellon and is built by General Electric. It has a peak rate of 100 million floating-point operations per second and can compute weight adjustments for back-propagation networks at a rate of 20 million connections per second.)

Once it is trained, ALVINN can accurately drive the NAVLAB vehicle at about 3½ miles per hour along a path through a wooded area adjoining the Carnegie Mellon campus, under a variety of weather and lighting conditions. This speed is nearly twice as fast as that achieved by non-neural-network algorithms running on the same vehicle. Part of the reason for this is that the forward pass of a back-propagation network can be computed quickly. It takes about 200 milliseconds on the Sun-3/160 workstation installed on the NAVLAB.

The hidden-layer representations ALVINN develops are interesting. When trained on roads of a fixed width, the network chooses a representation in which hidden units act as detectors for complete roads at various positions and orientations. When trained on roads of variable

continued

Photo 1: *The NAVLAB autonomous navigation test-bed vehicle and the road used for trial runs.*

1995: 2797/2849 miles (98.2%)

NO HANDS ACROSS AMERICA

Carnegie Mellon University • Delco Electronics • AssistWare Technology

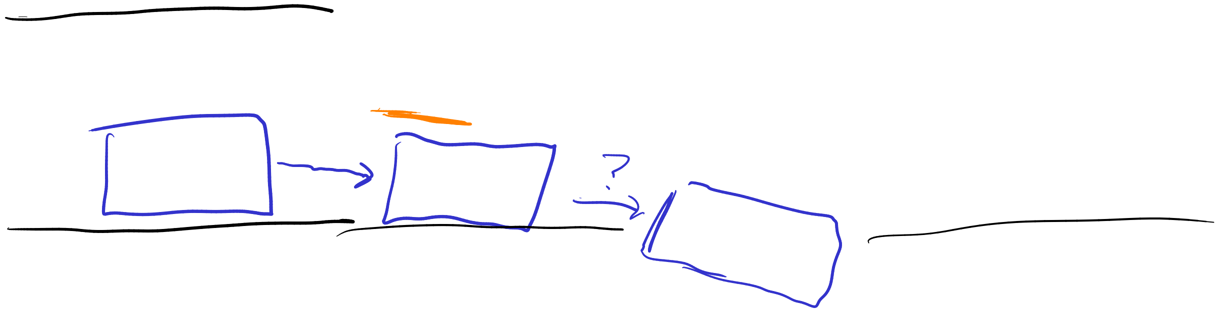
Behavioral Cloning

Behavioral Cloning

$$\underset{\theta}{\text{maximize}} \prod_{(s,a)\in D} \pi_\theta(a \mid s)$$

states

expert action



Behavioral Cloning

$$\underset{\theta}{\text{maximize}} \prod_{(s,a)\in D} \pi_\theta(a \mid s)$$

Problem: Cascading Errors

How did ALVINN do it?

3.2. *TRAINING "ON-THE-FLY" WITH REAL DATA*

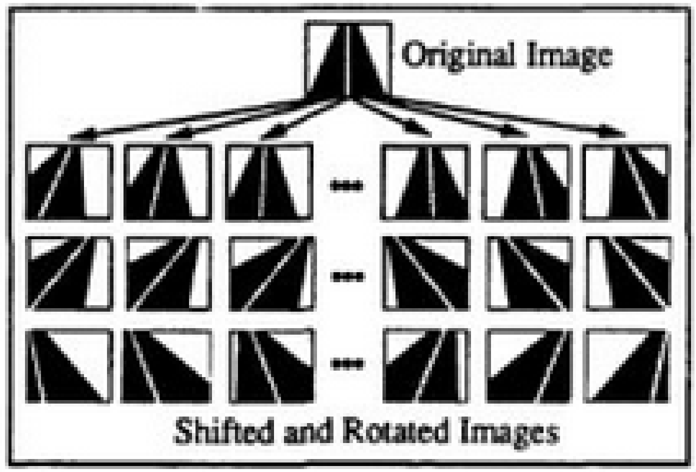


Figure 3.4: The single original video image is shifted and rotated to create multiple training exemplars in which the vehicle appears to be at different locations relative to the road.

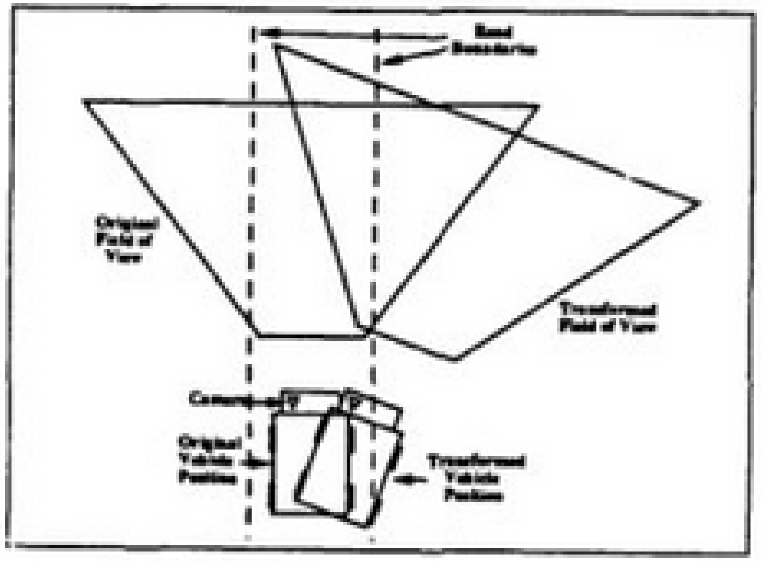
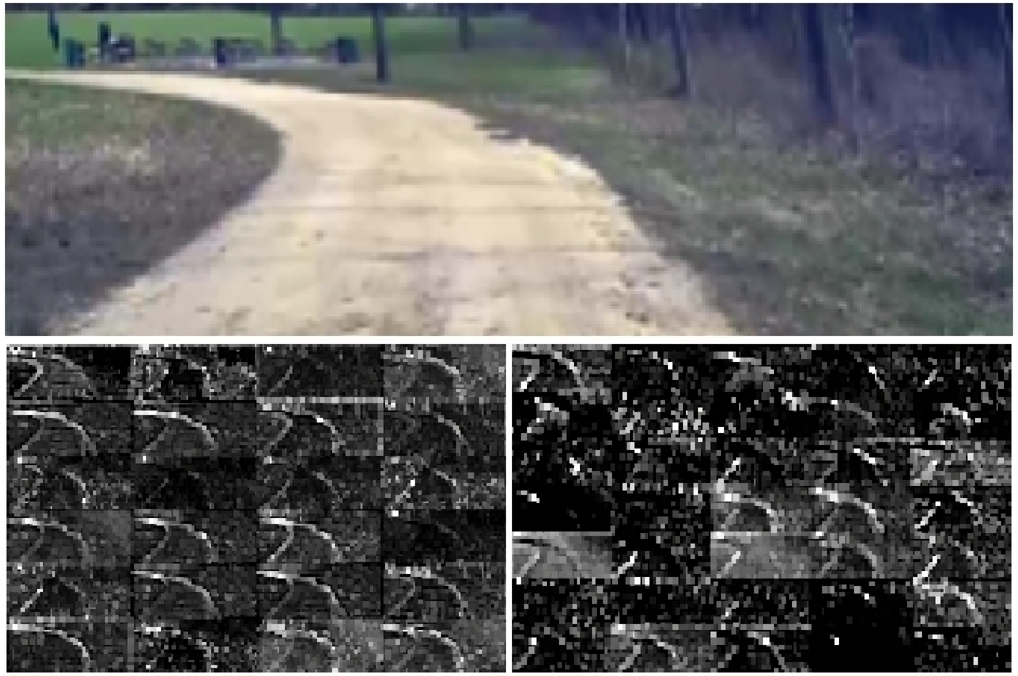


Figure 3.5: An aerial view of the vehicle at two different positions, with the corresponding sensor fields of view. To simulate the image transformation that would result from such a change in position and orientation of the vehicle, the overlap between the two field of view trapezoids is computed and used to direct resampling of the original image.

How did NVIDIA do it in 2016?



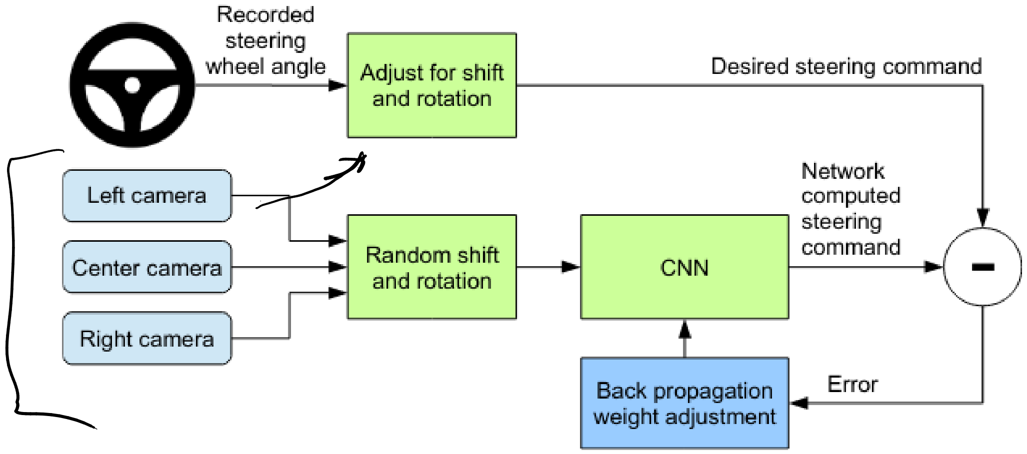


Figure 2: Training the neural network.

Dataset Aggregation (DAgger)

```
function optimize(M::DataSetAggregation, D, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, πE, πθ = M.d, M.b, M.πE, M.πθ
    θ = optimize(bc, D, θ)
    for k in 2:k_max
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = rand(πθ(θ, s))
                s = rand(𝒫.T(s, a))
            end
        end
        θ = optimize(bc, D, θ)
    end
    return θ
end
```

Dataset Aggregation (DAgger)

```julia
function optimize(M::DataSetAggregation, D, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, πE, πθ = M.d, M.b, M.πE, M.πθ
    θ = optimize(bc, D, θ)
    for k in 2:k_max
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = rand(πθ(θ, s))
                s = rand(𝒫.T(s, a))
            end
        end
        θ = optimize(bc, D, θ)
    end
    return θ
end
```

rollout

Dataset Aggregation (DAgger)

```
function optimize(M::DataSetAggregation, D, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, πE, πθ = M.d, M.b, M.πE, M.πθ
    θ = optimize(bc, D, θ)
    for k in 2:k_max
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = rand(πθ(θ, s))
                s = rand(𝒫.T(s, a))
            end
        end
        θ = optimize(bc, D, θ)
    end
    return θ
end
```

Gather from expert

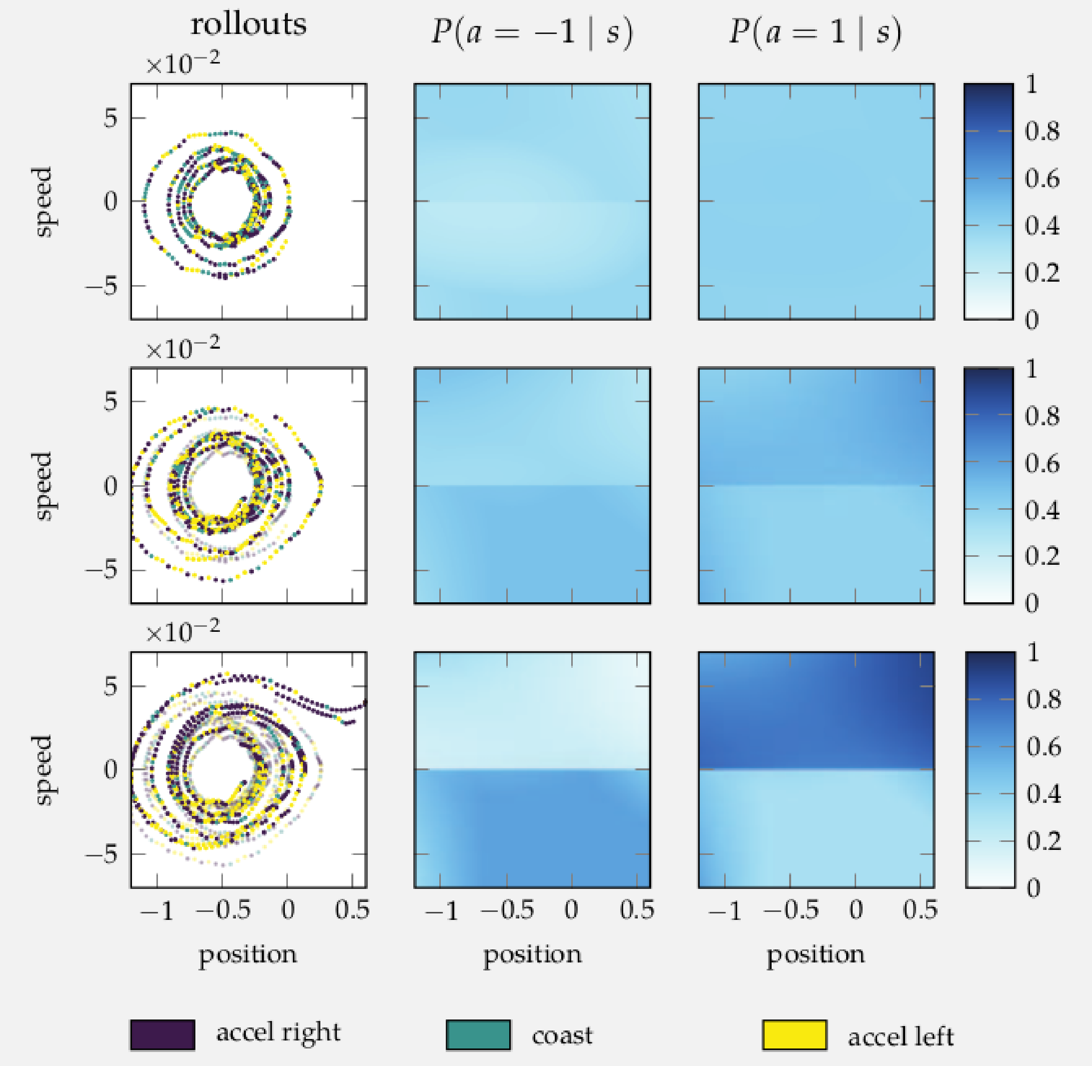
rollout

Dataset Aggregation (DAgger)

```
function optimize(M::DataSetAggregation, D, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, πE, πθ = M.d, M.b, M.πE, M.πθ
    θ = optimize(bc, D, θ)
    for k in 2:k_max
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = rand(πθ(θ, s))
                s = rand(𝒫.T(s, a))
            end
        end
        θ = optimize(bc, D, θ)
    end
    return θ
end
```

Gather from expert

rollout



Stochastic Mixing Iterative Learning (SMILe)

```
function optimize(M::SMILe, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, β, πE, πθ = M.d, M.b, M.β, M.πE, M.πθ
    𝒜, T = 𝒫.𝒜, 𝒫.T
    θs = []
    π = s -> πE(s)
    for k in 1:k_max
        # execute latest π to get new data set D
        D = []
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = π(s)
                s = rand(T(s, a))
            end
        end
        # train new policy classifier
        θ = optimize(bc, D, θ)
        push!(θs, θ)
        # compute a new policy mixture
        Pπ = Categorical(normalize([(1-β)^(i-1) for i in 1:k],1))
        π = s -> begin
            if rand() < (1-β)^(k-1)
                return πE(s)
            else
                return rand(Categorical(πθ(θs[rand(Pπ)], s)))
            end
        end
    end
    Ps = normalize([(1-β)^(i-1) for i in 1:k_max],1)
    return Ps, θs
end
```

Stochastic Mixing Iterative Learning (SMILe)

```julia
function optimize(M::SMILe, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, β, πE, πθ = M.d, M.b, M.β, M.πE, M.πθ
    𝒜, T = 𝒫.𝒜, 𝒫.T
    θs = []
    π = s -> πE(s)
    for k in 1:k_max
        # execute latest π to get new data set D
        D = [] ← reset D
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = π(s)
                s = rand(T(s, a))
            end
        end
        # train new policy classifier
        θ = optimize(bc, D, θ)
        push!(θs, θ)
        # compute a new policy mixture
        Pπ = Categorical(normalize([(1-β)^(i-1) for i in 1:k],1))
        π = s -> begin
            if rand() < (1-β)^(k-1)
                return πE(s)
            else
                return rand(Categorical(πθ(θs[rand(Pπ)], s)))
            end
        end
    end
    Ps = normalize([(1-β)^(i-1) for i in 1:k_max],1)
    return Ps, θs
end
```

Stochastic Mixing Iterative Learning (SMILe)

```
function optimize(M::SMILe, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, β, πE, πθ = M.d, M.b, M.β, M.πE, M.πθ
    𝒜, T = 𝒫.𝒜, 𝒫.T
    θs = []
    π = s -> πE(s)
    for k in 1:k_max
        # execute latest π to get new data set D
        D = []          ← reset D
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = π(s)
                s = rand(T(s, a))
            end
        end
        # train new policy classifier
        θ = optimize(bc, D, θ)
        push!(θs, θ)
        # compute a new policy mixture
        Pπ = Categorical(normalize([(1-β)^(i-1) for i in 1:k],1))
        π = s -> begin
            if rand() < (1-β)^(k-1)
                return πE(s)
            else
                return rand(Categorical(πθ(θs[rand(Pπ)], s)))
            end
        end
    end
    Ps = normalize([(1-β)^(i-1) for i in 1:k_max],1)
    return Ps, θs
end
```

} Gather data

Stochastic Mixing Iterative Learning (SMILe)

```
function optimize(M::SMILe, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, β, πE, πθ = M.d, M.b, M.β, M.πE, M.πθ
    𝒜, T = 𝒫.𝒜, 𝒫.T
    θs = []
    π = s -> πE(s)
    for k in 1:k_max
        # execute latest π to get new data set D
        D = []
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = π(s)
                s = rand(T(s, a))
            end
        end
        # train new policy classifier
        θ = optimize(bc, D, θ)
        push!(θs, θ)
        # compute a new policy mixture
        Pπ = Categorical(normalize([(1-β)^(i-1) for i in 1:k],1))
        π = s -> begin
            if rand() < (1-β)^(k-1)
                return πE(s)
            else
                return rand(Categorical(πθ(θs[rand(Pπ)], s)))
            end
        end
    end
    Ps = normalize([(1-β)^(i-1) for i in 1:k_max],1)
    return Ps, θs
end
```

reset D

Gather data

train only on D

Stochastic Mixing Iterative Learning (SMILe)

```
function optimize(M::SMILe, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, β, πE, πθ = M.d, M.b, M.β, M.πE, M.πθ
    𝒜, T = 𝒫.𝒜, 𝒫.T
    θs = []
    π = s -> πE(s)
    for k in 1:k_max
        # execute latest π to get new data set D
        D = []
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = π(s)
                s = rand(T(s, a))
            end
        end
        # train new policy classifier
        θ = optimize(bc, D, θ)
        push!(θs, θ)
        # compute a new policy mixture
        Pπ = Categorical(normalize([(1-β)^(i-1) for i in 1:k],1))
        π = s -> begin
            if rand() < (1-β)^(k-1)
                return πE(s)
            else
                return rand(Categorical(πθ(θs[rand(Pπ)], s)))
            end
        end
    end
    Ps = normalize([(1-β)^(i-1) for i in 1:k_max],1)
    return Ps, θs
end
```

reset D

Gather data

train only on D

Mix Policies

Stochastic Mixing Iterative Learning (SMILe)

```julia
function optimize(M::SMILe, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, β, πE, πθ = M.d, M.b, M.β, M.πE, M.πθ
    𝒜, T = 𝒫.𝒜, 𝒫.T
    θs = []
    π = s -> πE(s)
    for k in 1:k_max
        # execute latest π to get new data set D
        D = []
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = π(s)
                s = rand(T(s, a))
            end
        end
        # train new policy classifier
        θ = optimize(bc, D, θ)
        push!(θs, θ)
        # compute a new policy mixture
        Pπ = Categorical(normalize([(1-β)^(i-1) for i in 1:k],1))
        π = s -> begin
            if rand() < (1-β)^(k-1)
                return πE(s)
            else
                return rand(Categorical(πθ(θs[rand(Pπ)], s)))
            end
        end
    end
    Ps = normalize([(1-β)^(i-1) for i in 1:k_max],1)
    return Ps, θs
end
```

reset D

Gather data

train only on D

Mix Policies

$$(1-\beta)^k$$

Stochastic Mixing Iterative Learning (SMILe)

```julia
function optimize(M::SMILe, θ)
    𝒫, bc, k_max, m = M.𝒫, M.bc, M.k_max, M.m
    d, b, β, πE, πθ = M.d, M.b, M.β, M.πE, M.πθ
    𝒜, T = 𝒫.𝒜, 𝒫.T
    θs = []
    π = s -> πE(s)
    for k in 1:k_max
        # execute latest π to get new data set D
        D = []
        for i in 1:m
            s = rand(b)
            for j in 1:d
                push!(D, (s, πE(s)))
                a = π(s)
                s = rand(T(s, a))
            end
        end
        # train new policy classifier
        θ = optimize(bc, D, θ)
        push!(θs, θ)
        # compute a new policy mixture
        Pπ = Categorical(normalize([(1-β)^(i-1) for i in 1:k],1))
        π = s -> begin
            if rand() < (1-β)^(k-1)
                return πE(s)
            else
                return rand(Categorical(πθ(θs[rand(Pπ)], s)))
            end
        end
    end
    Ps = normalize([(1-β)^(i-1) for i in 1:k_max],1)
    return Ps, θs
end
```

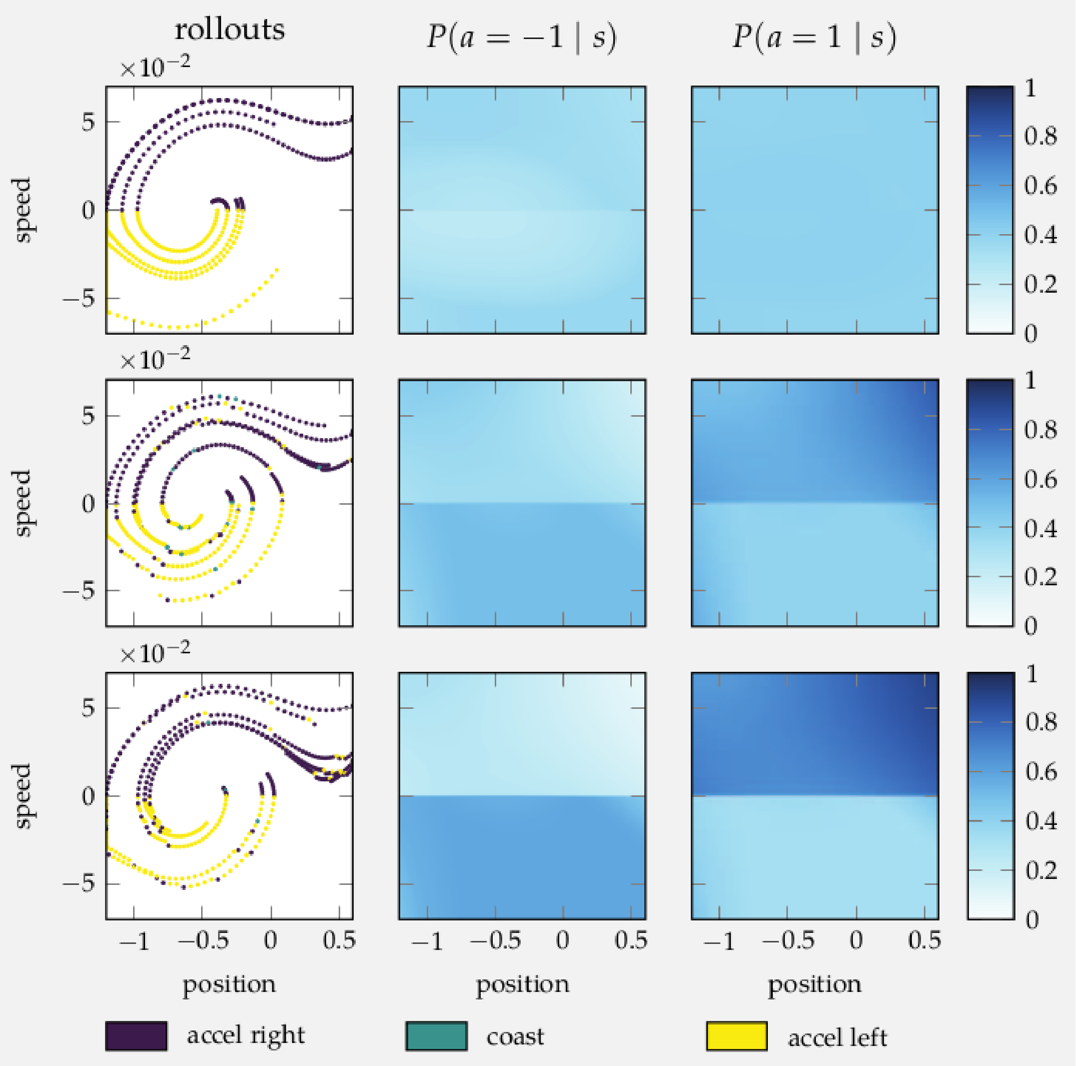
reset D

Gather data

train only on D

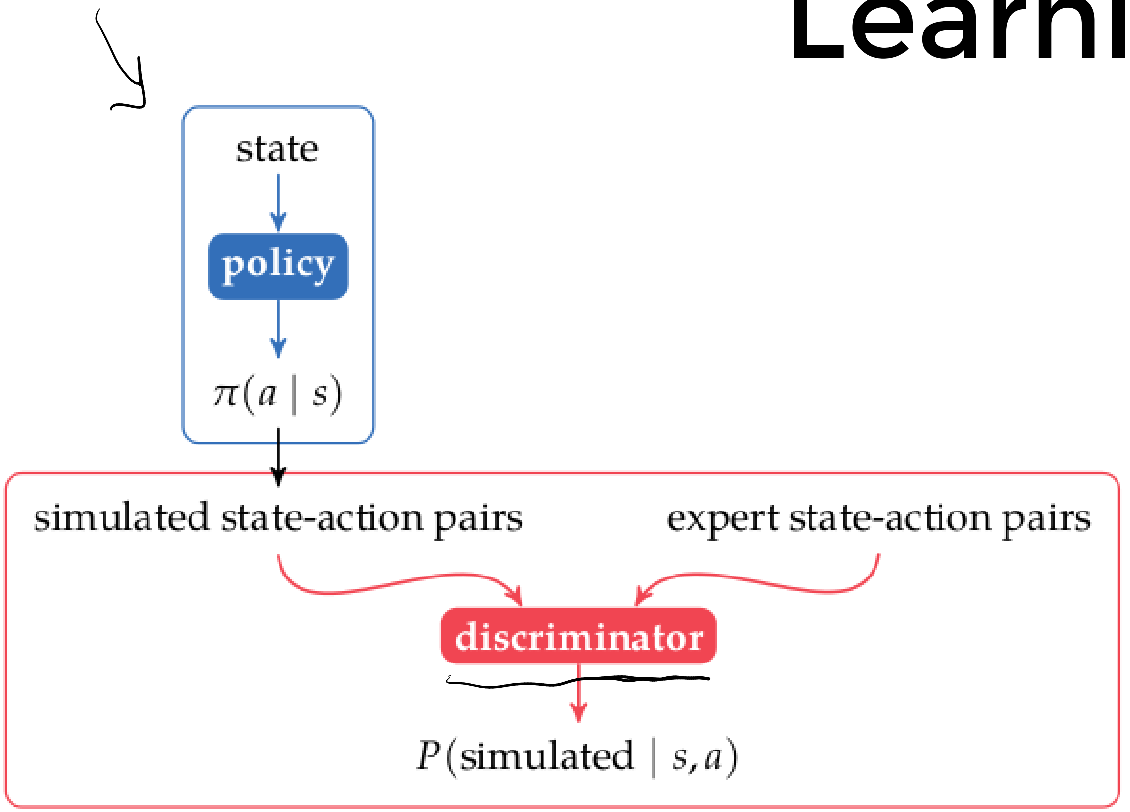
Mix Policies

$$(1-\beta)^k$$

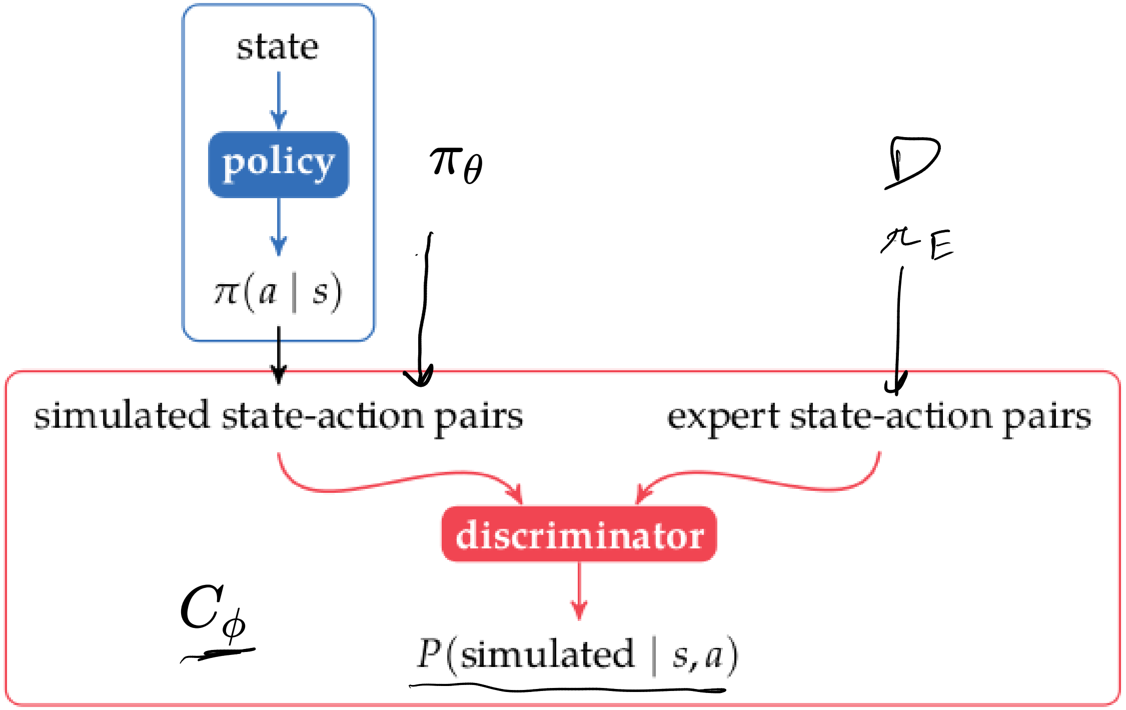


Generative Adversarial Imitation Learning (GAIL)

Generative Adversarial Imitation Learning (GAIL)

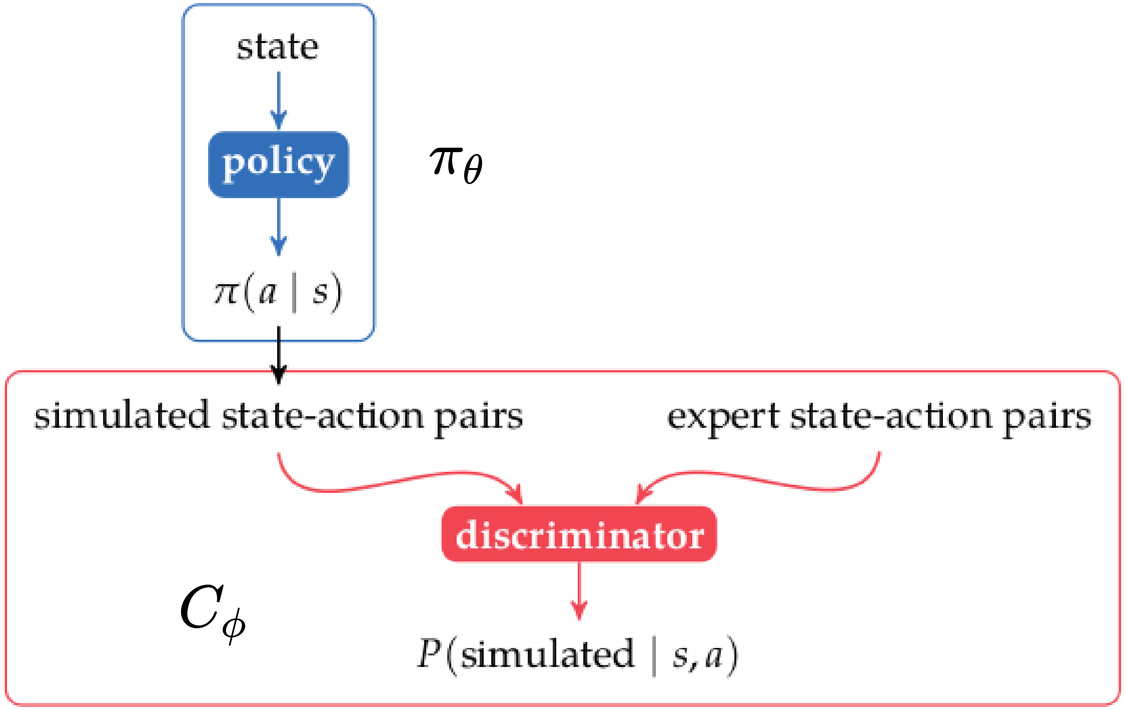


Generative Adversarial Imitation Learning (GAIL)



$$\max_{\phi} \min_{\theta} \mathbb{E}_{(s,a)\sim\pi_\theta}\left[\log(C_\phi(s,a))\right] + \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\log(1 - C_\phi(s,a))\right]$$

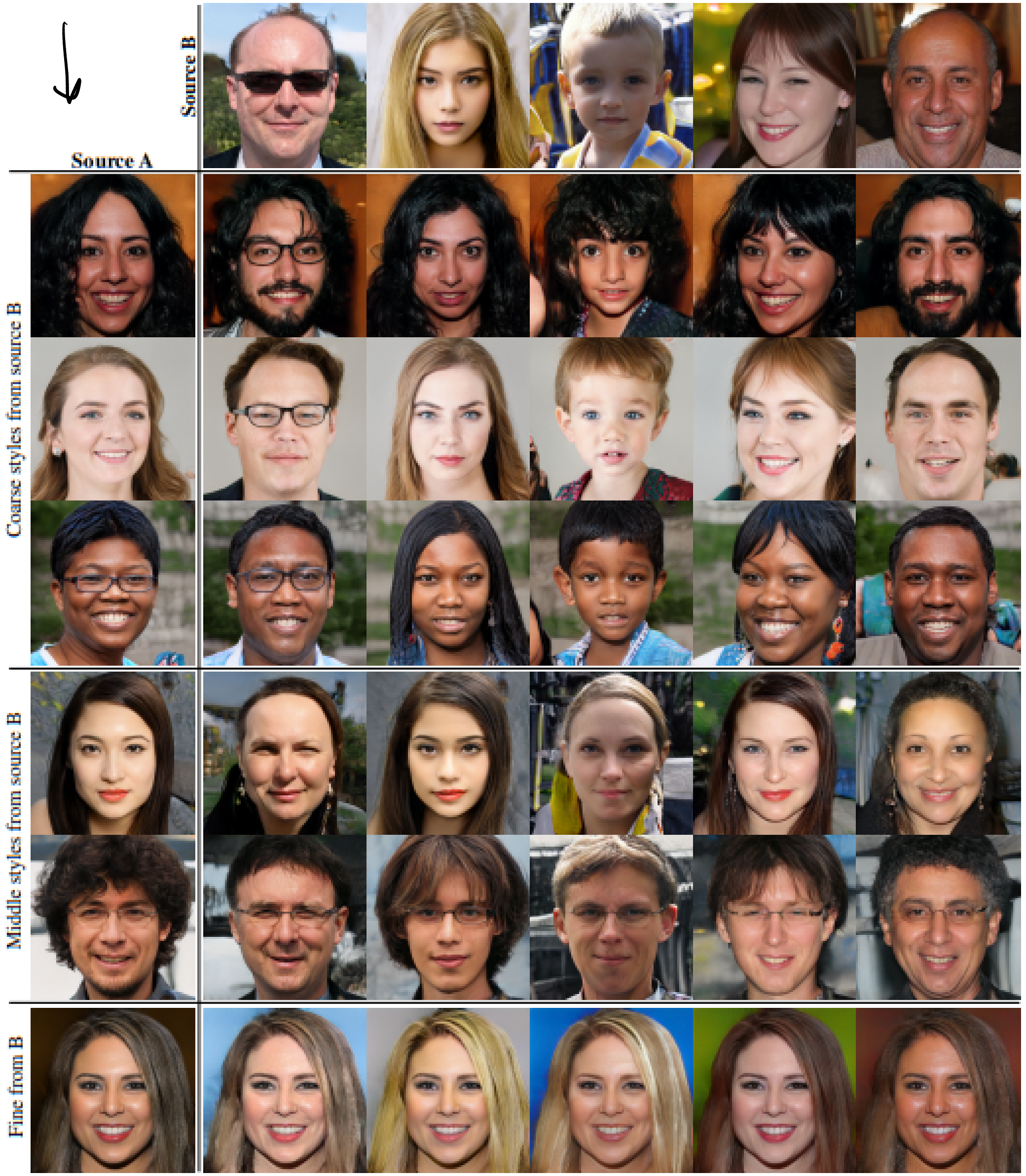
Generative Adversarial Imitation Learning (GAIL)



GANs are frighteningly good at generating believable synthetic things

$$\max_{\phi} \min_{\theta} \mathbb{E}_{(s,a)\sim\pi_\theta}\left[\log(C_\phi(s,a))\right] + \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\log(1 - C_\phi(s,a))\right]$$

Generative Adversarial Imitation ...g (GAIL)



simula

$\max_{\phi} \min_{\theta}$

GANs are frighteningly good at generating believable synthetic things

Inverse Reinforcement Learning

Inverse Reinforcement Learning

What if we know the dynamics, but not the reward?

Inverse Reinforcement Learning

What if we know the dynamics, but not the reward?

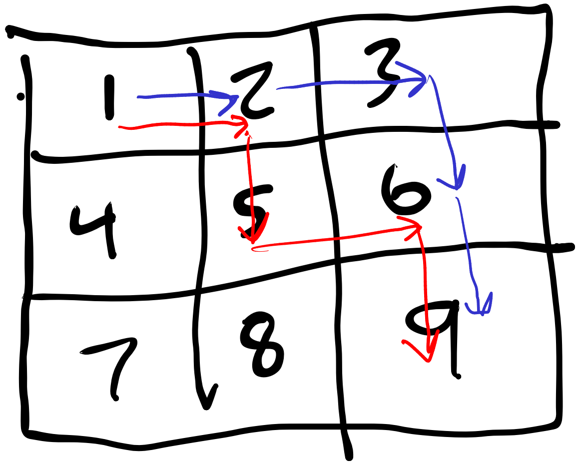
	Reinforcement Learning	**Inverse Reinforcement Learning**
Input		
Output		

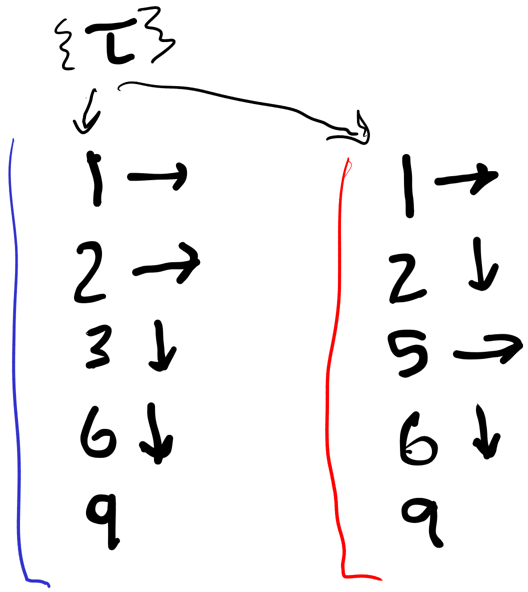
Inverse Reinforcement Learning

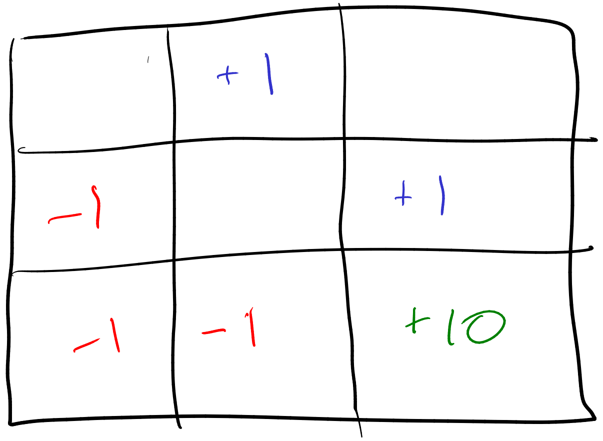
What if we know the dynamics, but not the reward?

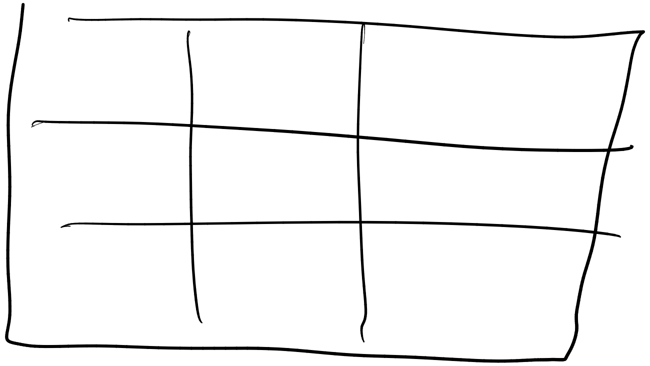
	Reinforcement Learning	Inverse Reinforcement Learning
Input	Environment $(S, A, \underline{T}, \underline{R})$	$\underline{S}, \underline{A}, \underline{T}, \{\tau\}$ trajectories from expert
Output	π^*	R

Exercise









What is the reward function?

IRL is an underspecified

Maximum Margin Inverse Reinforcement Learning

Maximum Margin Inverse Reinforcement Learning

$$R_{\boldsymbol{\phi}}(s,a) = \underline{\boldsymbol{\phi}}^\top \underbrace{\boldsymbol{\beta}(s,a)}_{\text{features}}$$

Maximum Margin Inverse Reinforcement Learning

$$R_{\boldsymbol{\phi}}(s,a) = \boldsymbol{\phi}^\top \boldsymbol{\beta}(s,a)$$

$$\beta(s,a) \in \{0,1\}^n$$

e.g. $\beta^i = \begin{cases} 1 & \text{if } s = s^i \\ 0 & \text{o.w.} \end{cases}$

$\beta^i = \begin{cases} 1 & \text{if } s \text{ in leftmost column} \\ & \end{cases}$

$\beta = \begin{cases} 1 & \text{if within 2m of centerline} \\ & \end{cases}$

Maximum Margin Inverse Reinforcement Learning

$$R_{\boldsymbol{\phi}}(s,a) = \boldsymbol{\phi}^\top \boldsymbol{\beta}(s,a)$$

$$\beta(s,a) \in \{0,1\}^n$$

$$\|\boldsymbol{\phi}\|_2 \leq 1$$

Maximum Margin Inverse Reinforcement Learning

$$R_{\boldsymbol{\phi}}(s, a) = \boldsymbol{\phi}^\top \boldsymbol{\beta}(s, a)$$

$$\beta(s, a) \in \{0, 1\}^n$$

$$\|\boldsymbol{\phi}\|_2 \leq 1$$

$$\begin{aligned}
\mathop{\mathbb{E}}_{s \sim b}[U(s)] &= \mathbb{E}_\tau\left[\sum_{k=1}^{d} \gamma^{k-1} \underline{R_{\boldsymbol{\phi}}(s^{(k)}, a^{(k)})}\right] \\
&= \mathbb{E}_\tau\left[\sum_{k=1}^{d} \gamma^{k-1} \boldsymbol{\phi}^\top \boldsymbol{\beta}(s^{(k)}, a^{(k)})\right] \\
&= \boldsymbol{\phi}^\top \underbrace{\left(\mathbb{E}_\tau\left[\sum_{k=1}^{d} \gamma^{k-1} \boldsymbol{\beta}(s^{(k)}, a^{(k)})\right]\right)} \\
&= \boldsymbol{\phi}^\top \boldsymbol{\mu}_\pi
\end{aligned}$$

discounted expected value of β

Maximum Margin Inverse Reinforcement Learning

$$R_{\boldsymbol{\phi}}(s,a) = \boldsymbol{\phi}^\top \boldsymbol{\beta}(s,a)$$

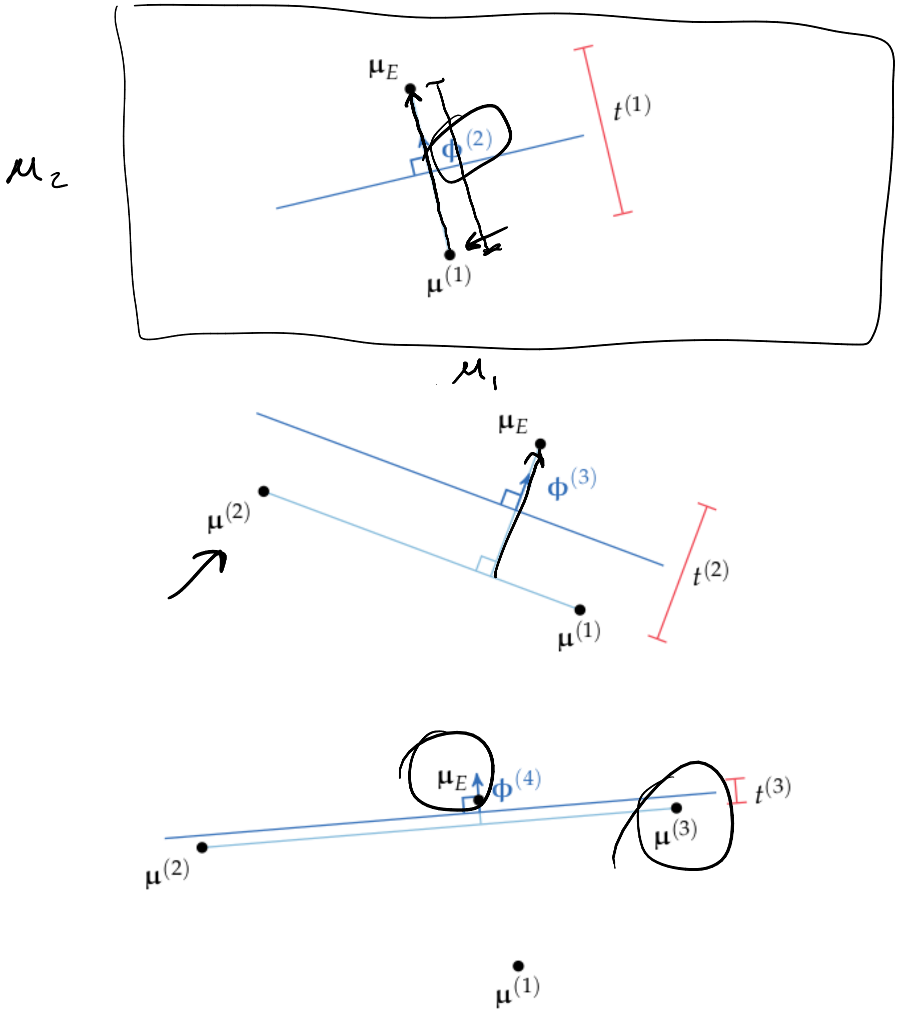
$$\beta(s,a) \in \{0,1\}^n$$

$$\|\boldsymbol{\phi}\|_2 \le 1$$

$$\begin{aligned}
\underset{t,\boldsymbol{\phi}}{\text{maximize}} \quad & t \\
\text{subject to} \quad & \boldsymbol{\phi}^\top \boldsymbol{\mu}_E \ge \boldsymbol{\phi}^\top \boldsymbol{\mu}^{(i)} + t \;\; \text{for} \;\; i = 1, \ldots, k-1 \\
& \|\boldsymbol{\phi}\|_2 \le 1
\end{aligned}$$

$$\begin{aligned}
\mathop{\mathbb{E}}_{s\sim b}[U(s)] &= \mathbb{E}_\tau\left[\sum_{k=1}^{d} \gamma^{k-1} R_{\boldsymbol{\phi}}(s^{(k)}, a^{(k)})\right] \\
&= \mathbb{E}_\tau\left[\sum_{k=1}^{d} \gamma^{k-1} \boldsymbol{\phi}^\top \boldsymbol{\beta}(s^{(k)}, a^{(k)})\right] \\
&= \boldsymbol{\phi}^\top \left(\mathbb{E}_\tau\left[\sum_{k=1}^{d} \gamma^{k-1} \boldsymbol{\beta}(s^{(k)}, a^{(k)})\right]\right) \\
&= \boldsymbol{\phi}^\top \boldsymbol{\mu}_\pi
\end{aligned}$$

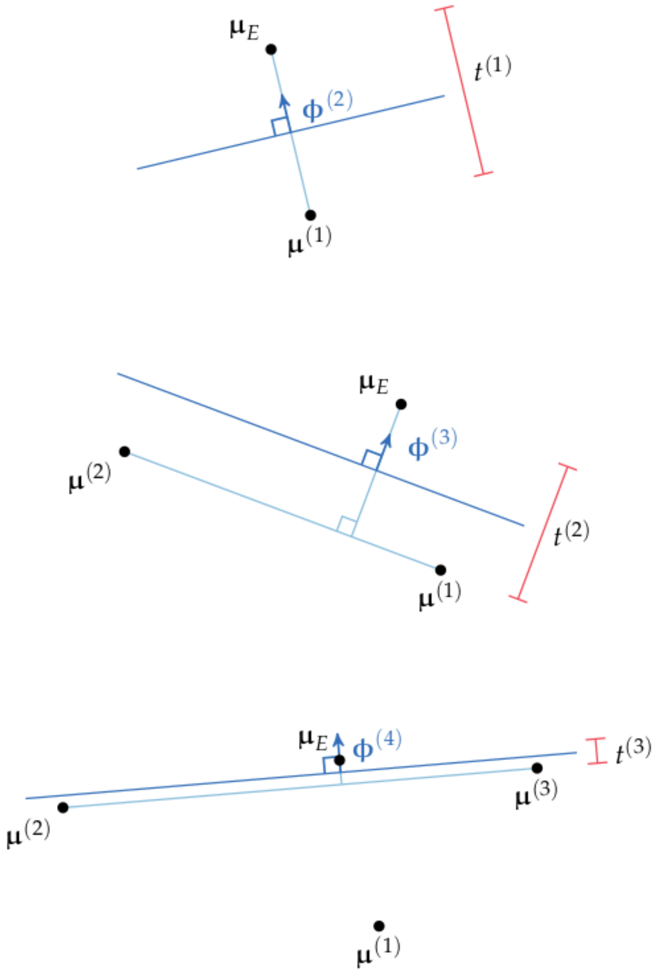
Maximum Margin Inverse Reinforcement Learning



Choose random ϕ
optimize π^*_ϕ
estimate μ for π^*_ϕ

$$
\begin{aligned}
\underset{t,\boldsymbol{\phi}}{\text{maximize}} \quad & t \\
\text{subject to} \quad & \boldsymbol{\phi}^\top \boldsymbol{\mu}_E \geq \boldsymbol{\phi}^\top \boldsymbol{\mu}^{(i)} + t \ \text{ for } i = 1, \ldots, k-1 \\
& \|\boldsymbol{\phi}\|_2 \leq 1
\end{aligned}
$$

Maximum Margin Inverse Reinforcement Learning



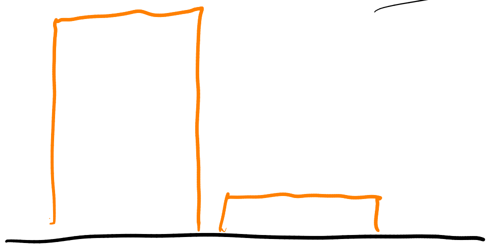
$$\begin{aligned} \underset{t,\phi}{\text{maximize}} \quad & t \\ \text{subject to} \quad & \phi^\top \mu_E \geq \phi^\top \mu^{(i)} + t \ \text{ for } \ i = 1, \ldots, k-1 \\ & \|\phi\|_2 \leq 1 \end{aligned}$$

$$\begin{aligned} \underset{\lambda}{\text{minimize}} \quad & \|\mu_E - \mu_\lambda\|_2 \\ \text{subject to} \quad & \lambda \geq 0 \\ & \|\lambda\|_1 = 1 \end{aligned}$$

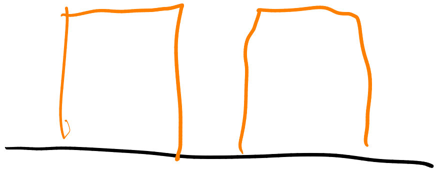
Principle of Maximum Entropy

$\tau \quad \pi_{\hat{R}}$

$$H(X) = -\sum_x P(x) \log P(x)$$



Low entropy



high entropy

Maximum Entropy Inverse Reinforcement Learning

Maximum Entropy Inverse Reinforcement Learning

Least informative trajectory distribution

Maximum Entropy Inverse Reinforcement Learning

Least informative trajectory distribution

$$P_{\boldsymbol{\phi}}(\tau) = \frac{1}{\mathrm{Z}(\boldsymbol{\phi})} \exp(R_{\boldsymbol{\phi}}(\tau))$$

Maximum Entropy Inverse Reinforcement Learning

Least informative trajectory distribution

$$P_{\phi}(\tau) = \frac{1}{Z(\phi)} \exp(R_{\phi}(\tau))$$

$$Z(\phi) = \sum_{\tau} \exp(R_{\phi}(\tau))$$

Maximum Entropy Inverse Reinforcement Learning

Least informative trajectory distribution

$$P_{\boldsymbol{\phi}}(\tau) = \frac{1}{Z(\boldsymbol{\phi})} \exp(R_{\boldsymbol{\phi}}(\tau))$$

$$Z(\boldsymbol{\phi}) = \sum_{\tau} \exp(R_{\boldsymbol{\phi}}(\tau))$$

$$\max_{\boldsymbol{\phi}} f(\boldsymbol{\phi}) = \max_{\boldsymbol{\phi}} \underbrace{\sum_{\tau \in \mathcal{D}}}_{} \overset{P_{\phi}(\tau)}{\underline{\log P_{\boldsymbol{\phi}}(\tau)}}$$

Maximum Entropy Inverse Reinforcement Learning

Maximum Entropy Inverse Reinforcement Learning

$$\max_{\phi} f(\phi) = \max_{\phi} \sum_{\tau \in \mathcal{D}} \log P_{\phi}(\tau)$$

Maximum Entropy Inverse Reinforcement Learning

$$\max_{\phi} f(\phi) = \max_{\phi} \sum_{\tau \in \mathcal{D}} \log P_{\phi}(\tau)$$

$$\begin{aligned}
f(\phi) &= \sum_{\tau \in \mathcal{D}} \log\left(\frac{1}{Z(\phi)} \exp(R_{\phi}(\tau))\right) \\
&= \left(\sum_{\tau \in \mathcal{D}} R_{\phi}(\tau)\right) - |\mathcal{D}| \log Z(\phi) \\
&= \left(\sum_{\tau \in \mathcal{D}} R_{\phi}(\tau)\right) - |\mathcal{D}| \log \sum_{\tau} \exp(R_{\phi}(\tau))
\end{aligned}$$

Maximum Entropy Inverse Reinforcement Learning

$$\max_{\boldsymbol{\phi}} f(\boldsymbol{\phi}) = \max_{\boldsymbol{\phi}} \sum_{\tau \in \mathcal{D}} \log P_{\boldsymbol{\phi}}(\tau)$$

$$\begin{aligned}
f(\boldsymbol{\phi}) &= \sum_{\tau \in \mathcal{D}} \log \frac{1}{Z(\boldsymbol{\phi})} \exp(R_{\boldsymbol{\phi}}(\tau)) \\
&= \left(\sum_{\tau \in \mathcal{D}} R_{\boldsymbol{\phi}}(\tau) \right) - |\mathcal{D}| \log Z(\boldsymbol{\phi}) \\
&= \left(\sum_{\tau \in \mathcal{D}} R_{\boldsymbol{\phi}}(\tau) \right) - |\mathcal{D}| \log \sum_{\tau} \exp(R_{\boldsymbol{\phi}}(\tau))
\end{aligned}$$

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} f &= \left(\sum_{\tau \in \mathcal{D}} \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) \right) - \underbrace{\frac{|\mathcal{D}|}{\sum_{\tau} \exp(R_{\boldsymbol{\phi}}(\tau))} \sum_{\tau} \exp(R_{\boldsymbol{\phi}}(\tau))}\nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) && (18.15) \\
&= \left(\sum_{\tau \in \mathcal{D}} \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) \right) - |\mathcal{D}| \sum_{\tau} P_{\boldsymbol{\phi}}(\tau) \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) && (18.16) \\
&= \left(\sum_{\tau \in \mathcal{D}} \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(\tau) \right) - |\mathcal{D}| \overbrace{\sum_{s} b_{\gamma,\boldsymbol{\phi}}(s) \sum_{a} \pi_{\boldsymbol{\phi}}(a \mid s)} \nabla_{\boldsymbol{\phi}} R_{\boldsymbol{\phi}}(s, a) && (18.17)
\end{aligned}$$

Maximum Entropy Inverse Reinforcement Learning

$$\max_{\phi} f(\phi) = \max_{\phi} \sum_{\tau \in \mathcal{D}} \log P_{\phi}(\tau)$$

$$\begin{aligned} f(\phi) &= \sum_{\tau \in \mathcal{D}} \log \frac{1}{Z(\phi)} \exp(R_{\phi}(\tau)) \\ &= \left(\sum_{\tau \in \mathcal{D}} R_{\phi}(\tau)\right) - |\mathcal{D}| \log Z(\phi) \\ &= \left(\sum_{\tau \in \mathcal{D}} R_{\phi}(\tau)\right) - |\mathcal{D}| \log \sum_{\tau} \exp(R_{\phi}(\tau)) \end{aligned}$$

$$\nabla_{\phi} f = \left(\sum_{\tau \in \mathcal{D}} \nabla_{\phi} R_{\phi}(\tau)\right) - \frac{|\mathcal{D}|}{\sum_{\tau} \exp(R_{\phi}(\tau))} \sum_{\tau} \exp(R_{\phi}(\tau)) \nabla_{\phi} R_{\phi}(\tau) \tag{18.15}$$

$$= \left(\sum_{\tau \in \mathcal{D}} \nabla_{\phi} R_{\phi}(\tau)\right) - |\mathcal{D}| \sum_{\tau} P_{\phi}(\tau) \nabla_{\phi} R_{\phi}(\tau) \tag{18.16}$$

$$= \left(\sum_{\tau \in \mathcal{D}} \nabla_{\phi} R_{\phi}(\tau)\right) - |\mathcal{D}| \sum_{s} b_{\gamma,\phi}(s) \sum_{a} \pi_{\phi}(a \mid s) \nabla_{\phi} R_{\phi}(s,a) \tag{18.17}$$

Discounted visitation probability

Optimal policy under R_{ϕ}

→ optimize π_{ϕ} to max R_{ϕ}

π_{ϕ} → simulate to get $b_{\gamma,\phi}(s)$

Recap

Recap

- Behavioral cloning is supervised learning to match the actions of an expert

Recap

- Behavioral cloning is supervised learning to match the actions of an expert
- A critical problem is cascading errors, which can be addressed by gathering more data with DAgger or SMILe

Recap

- Behavioral cloning is supervised learning to match the actions of an expert
- A critical problem is cascading errors, which can be addressed by gathering more data with DAgger or SMILe
- Inverse reinforcement learning is the process of learning a reward functions from trajectories in an MDP

Recap

- Behavioral cloning is supervised learning to match the actions of an expert
- A critical problem is cascading errors, which can be addressed by gathering more data with DAgger or SMILe
- Inverse reinforcement learning is the process of learning a reward functions from trajectories in an MDP
- IRL is an underspecified problem

Recap

- Behavioral cloning is supervised learning to match the actions of an expert
- A critical problem is cascading errors, which can be addressed by gathering more data with DAgger or SMILe
- Inverse reinforcement learning is the process of learning a reward functions from trajectories in an MDP
- IRL is an underspecified problem
- Maximum entropy RL solves this problem by choosing the reward function that maximizes the entropy of the trajectories of the resulting policy