Lecture-18

(*) Let's say you want to build a diving car.

→ How do you model the state of the car:-

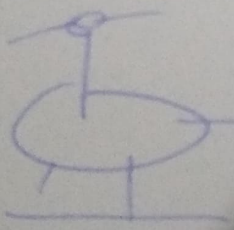
→ x, y → Position of the car

→ θ → Orientation of the car

→ $\dot{x}, \dot{y}, \dot{\theta}$ → Velocity in different direction.

← Six dimensional state space

(*) Let's you want to build a helicopter.



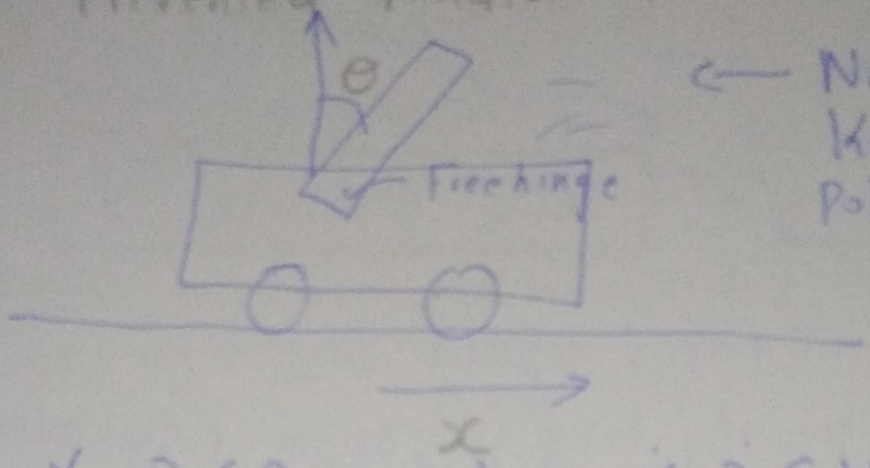
→ How do you model the state of the helicopter?

Roll, Pitch, Yaw

→ $x, y, z, \phi, \theta, \psi$ → Position

Position

→ Inverted Pendlum Problem.



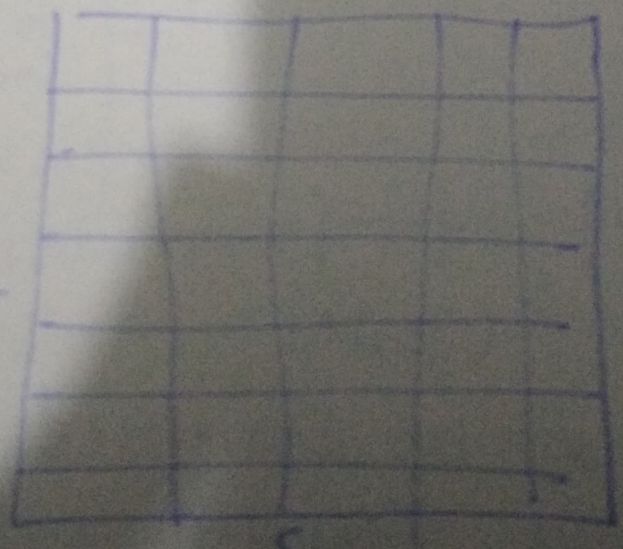
← Need to keep the Pole Balance.

X, θ (Position), $\dot{X}, \dot{\theta}$ (Velocity)

↳ State Space of Inverted Pendlum Problem

→ In above examples state space is Continous ↱ $S = \mathbb{R}^n$

④ Discretization:-

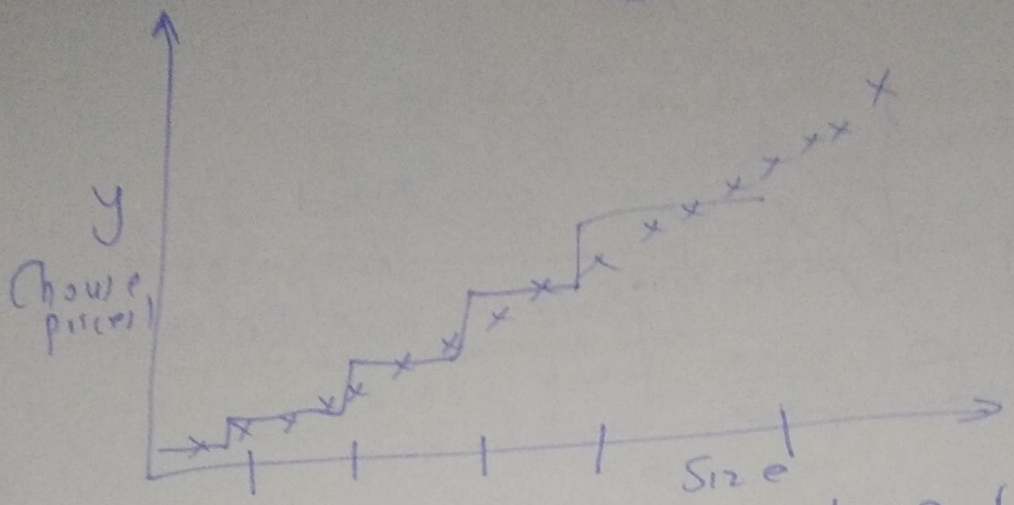


→ Two dimensional state space discretization

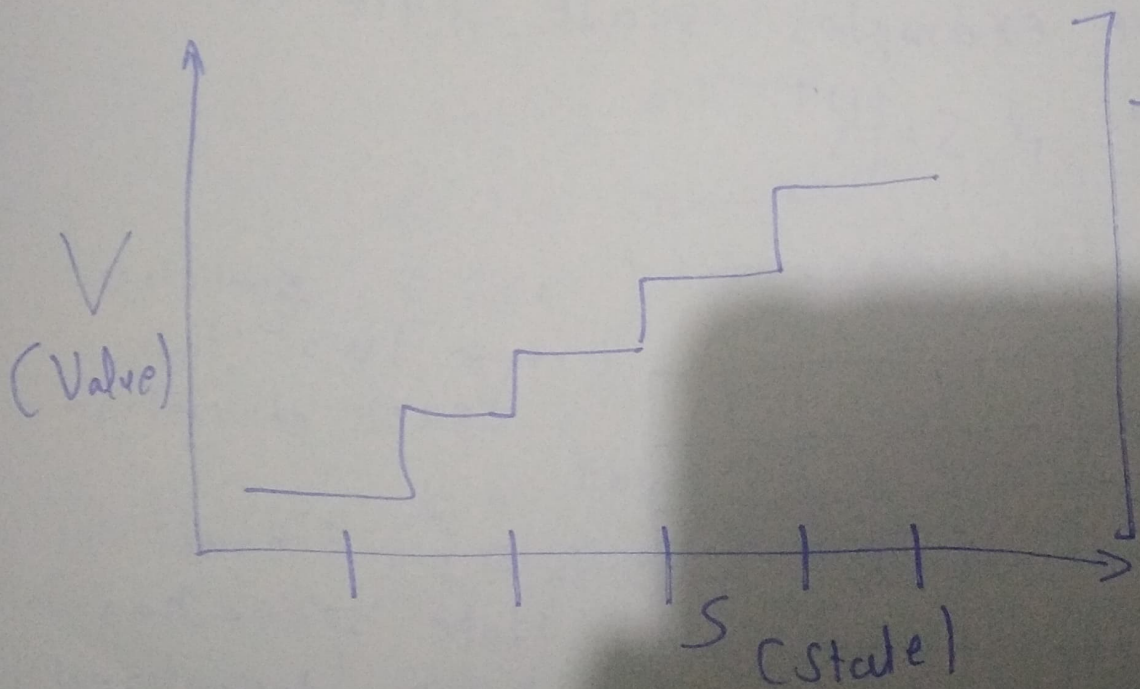
→ Continous state space has been discretized

⊙ Disadvantages: (Problems)

- Naive representation for V^*



↳ If we to discretize regression of size and house into five interval and fit separate regression into five intervals.



→ Same Analogy of above applies to reinforcement learning

↳ State space is discretized to map the value function.

↳ Function is not smooth and not a good representation.

- Curse of dimensionality:

⊛ $S = \mathbb{R}^n$ and discretize each dimension into k values, get

k^n discrete state.

→ Example of Curse of dimensionality:-

⊙ One of area where people have applied reinforcement learning is factory optimization:

⊙ So we have a factory with 100 machines in a factory.

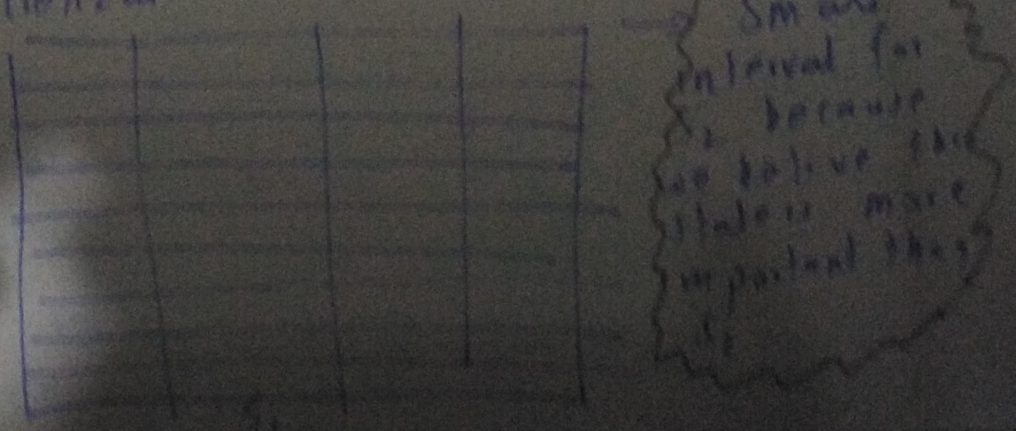
⊙ Every machine is doing something slightly different

⊙ 100 machines in k states

⊙ then total number of states is k^{100}

⊗ Discretization however is effective in small state space (2 dimension state space, 3 dimension state space)

⊗ As you reach four to six dimension choose your discretization more carefully:



- Seven to eight dimensions or higher are not suitable for discretization

- Approximate V^* directly without resorting to discretization

-In linear regression you say:

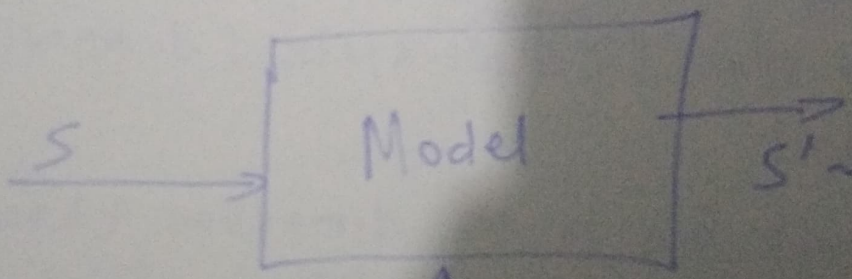
$X = \Theta^T y$ $\quad y = \Theta^T x$ or $\Theta^T \phi(x)$

$\phi(x)$ features of x $\quad \phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_{12} \\ x_1 x_2 \\ \vdots \end{bmatrix}$

$\rightarrow V(s) \approx \Theta^T \phi(s)$ (Fitted Value Iteration)

↳ where $\phi(s)$ is features of state

- Model (simulation) of MDP:



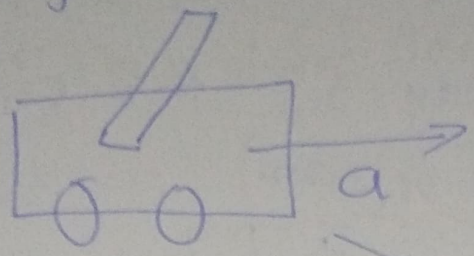
→ State is a row value vector

→ For now lets assume action space is discrete

→ Usually state space is greater than action space.

⊛ How to get a model?

① - Physics simulator



$$S = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix}$$

$$\dot{S} = \begin{pmatrix} \dot{x} \\ a - LB\cos(\theta)/m \\ \dot{\theta} \\ \beta \end{pmatrix}$$

→ Conventional Physics law.

$$S' = S + (\Delta t)\cdot \dot{S}$$

$$\Delta t = 0.1 \text{ seconds}$$

→ Simulation equations through physics.

② - Learn model from data:-

Human pilot deriving the helicopter and generating data!

$$S_0^{(1)} \xrightarrow{a_0^{(1)}} S_1^{(1)} \xrightarrow{a_1^{(1)}} S_2 \xrightarrow{a_2^{(1)}} S_3^{(1)} \longrightarrow \cdots \longrightarrow S_T^{(1)}$$

$$S_0^{(2)} \xrightarrow{a_0^{(2)}} \cdots \longrightarrow S_T^{(2)}$$

$$\vdots$$

$$S_0^{(m)} \longrightarrow \cdots$$

→ Fly helicopter

- Apply supervised learning to estimate

→ S_{t+1} as function of S_t, a_t

↳ Predicting S_{t+1} from S_t, a_t

Eg linear regression version:

$$S_{t+1} = AS_t + Ba_t$$

→ Not a terrible model

→ You can easily make it more complex

$A \in \mathbb{R}^{n \times n}$ ...

$$\min_{A,B} \sum_{i=1}^{m} \left\| S_{t+1} - (AS_t + Ba_t) \right\|^2$$

$$\min_{A,B} \sum_{i=1}^{m} \sum_{t=0}^{T} \left\| S_{t+1}^{(i)} - (AS_t^{(i)} + Ba_t^{(i)}) \right\|^2$$

Model (Model Based RL)

a) $S_{t+1} = AS_t + Ba_t$ (Deterministic)

or

b) $S_{t+1} = AS_t + Ba_t + \varepsilon_t$ (Stochastic)

where $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$

← You can choose either one!

$S_t \rightarrow$ [Model] $\rightarrow S_{t+1} \sim P_{sa}$ (Stochastic Model simulates the dynamics by introducing randomness)

a_t

(3) Once you have build a model, how do you approximate the value of function:

- Fitted value iteration:-

→ Choose features $\phi(s)$ of state s

$$V(s) = \theta^T(s)$$

$$\phi(s) = \begin{bmatrix} x \\ \dot{x} \\ x^2 \\ x\dot{x} \\ x\theta \\ \vdots \end{bmatrix}$$

→ Inverted Pendulum

→ Designing features that conveys how well the robot is doing (Very important task!)

→ ~~Main Algo~~

→ Fitted Value iteration Main Algorithm:

~ Previously (For discrete states)

→

⊙ Value iteration:

① $V(s) = R(s) + \gamma \max_a \sum_{s'} P_{sa}(s') V(s')$

Going to learn mapping from $s \longrightarrow V(s)$

$= R(s) + \gamma \max_a E_{s' \sim P_{sa}}[V(s')]$ hm

⊙ Lets generalize to fitted value iteration:

⊙ Sample $\{s^{(1)}, s^{(2)}, \ldots s^{(m)}\} \in S$ randomly

Initialize $\theta := 0$

• Repeat {

For $i = 1, \ldots m$ { ↙ Go over m states.

For each action $a \in A$ { ↙ Using model

Sample $s_1', s_2', \ldots s_k' \sim P_{s^{(i)}a}$

Set $q(a) = \frac{1}{k} \sum_{j=1}^{k} [R(s^{(i)}) + \gamma V(s_j')]$

}

$R(s) +$

$\approx E_{s' \sim P_{sa}}[V(s')]$

Set $y^{(i)} = \max_a q(a)$

}}

← Algorithm continues on next page.

→ $y^{(i)}$ is your estimate of value iteration

→ Orginal Value Iteration :-

$V(s) := y^{(i)}$

→ Not part of the algorithm : Explanation.

→ Fitted Value iteration :-

Want $V(s^{(i)}) \approx y^{(i)}$

$\downarrow$

$\theta^T \phi(s^{(i)})$

→ Final Step : of Algorithm

$$\theta := \arg\min_{\theta} \frac{1}{2} \sum_{i=1}^{m} \left(\theta^T(\phi(s^{(i)}) - y^{(i)}\right)^2$$

→ ~~algorithm ends here.~~ → Algorithm ends here.

→ Fitted Value Iteration gives an approximation to V^* implicitly defining π^* because

$\rightarrow \pi^*(s) = \arg\max_{a} E_{s' \sim P_{s'(a)}}[V(s')]$

⊙ For fitted Value Iteration :-

→ We Used $s_1', \ldots s_k' \sim P_{sa}$ to approximate expectation

→ Then how do you choose $\pi^*(s)$ given a state:-

Say model is:-

$$S_{t+1} = f(S_t, a_t) + \epsilon_t$$

$$e.g.\ (AS_t + Ba_t) + \epsilon_t)$$

→ For your deployment (runtime,

① Set $\epsilon_t = 0$ and $k = 1$

→ When in state s,

pick action

$$\pi^*(s) = \arg\max_a V(f(s,a))$$

↓ Simulator without noise.

$s' \sim P_{sa}$ but with deterministic model.