Model-based Reinforcement Learning

Peyman Noursalehi

Many slides are from or derived from Sergey Levine and Emma Brunskill

Categorizing RL algorithms

- Value based
  - No policy (implicit)
  - Value function
- Policy based
  - Policy
  - No value function
- Actor critic
  - Policy
  - Value function

- Model-based
  - Transition and reward model
- Model-free
  - No transition and no reward model (implicit)

Recap

Maximize the overall reward in a sequential problem

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right]$$

Where

$$\boxed{\pi_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)}$$

Model

Do we ever know the model?

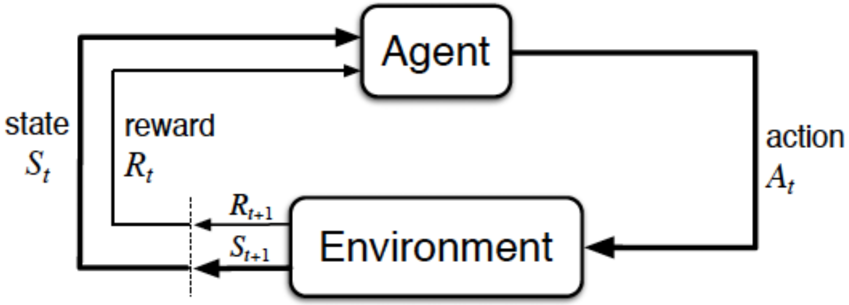
- Sometimes we know the true model
  - Chess, Go
  - Trajectory of a (small, short-ranged) rocket
  - Simulated environments
    - We cannot write down the math, but we have access to the source code

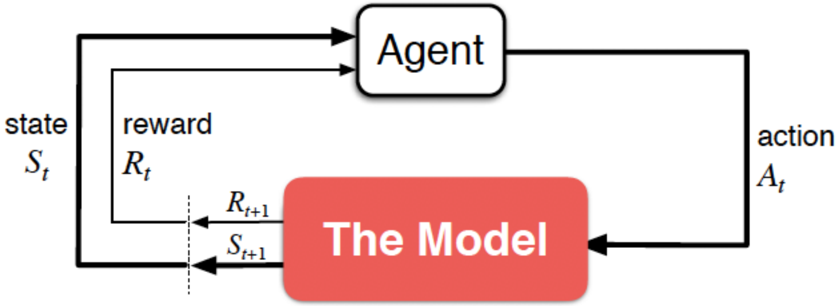
- Sometimes we know the form of the model, but not the parameters
  - System Identification

Does knowing the model help?

- Yes! We can plan ahead!

$$\text{Model} \xrightarrow{\text{Planning}} \text{Policy}$$



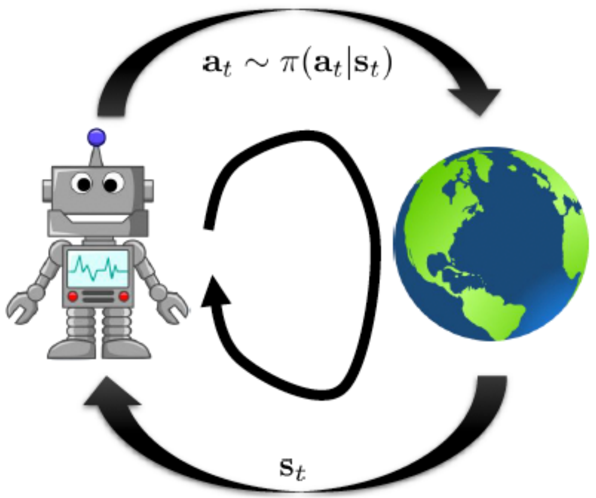


- Planning

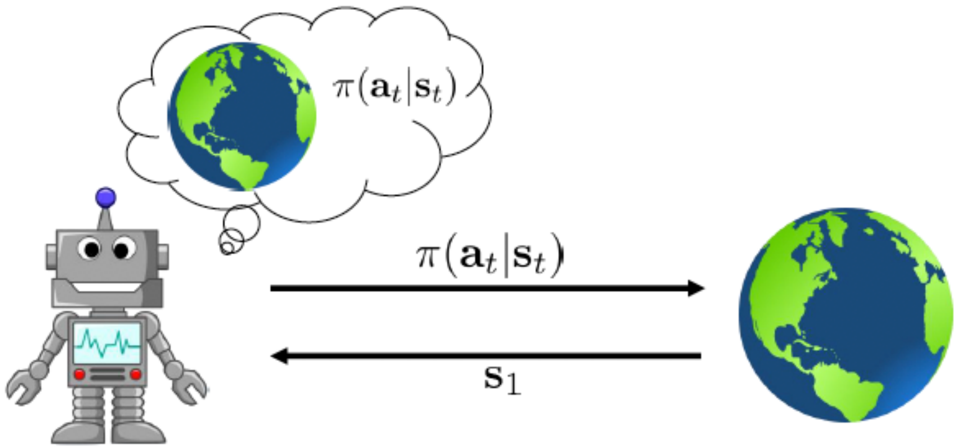
$$a_1, \ldots, a_T = arg \max_{a_1, \ldots, a_T} \sum_{t=1}^{t=T} r(s_t, a_t)$$

Open-loop VS closed-loop planning

Closed-loop



Open-loop



Is planning N-step ahead always helpful?

- Planning:

$$a_1, \ldots, a_T = arg \max_{a_1, \ldots, a_T} \sum_{t=1}^{t=T} r(s_t, a_t)$$

- Deterministic environment:

$$s_{t+1} = f(s_t, a_t)$$

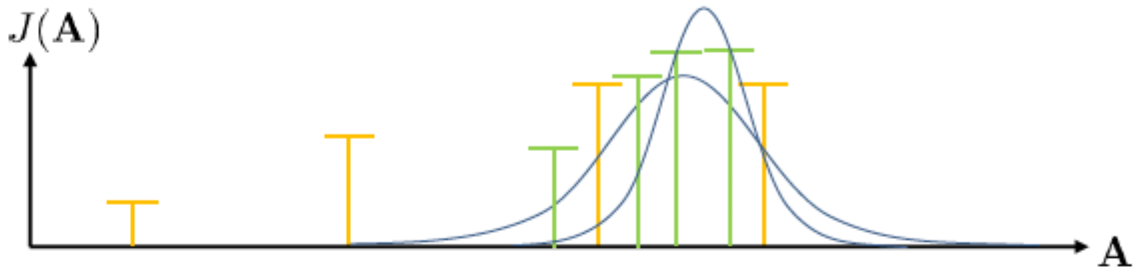
- Stochastic environment:

$$p(s_{t+1}|s_t, a_t)$$

How to plan ahead?

- First attempt: “Random Shooting”

1. Pick a set of action sequences according to some distribution
2. Evaluate the objective function for each set of actions
3. Keep the elite episodes (only keep those with rewards > threshold)
4. Refit the distribution based on the elite episodes



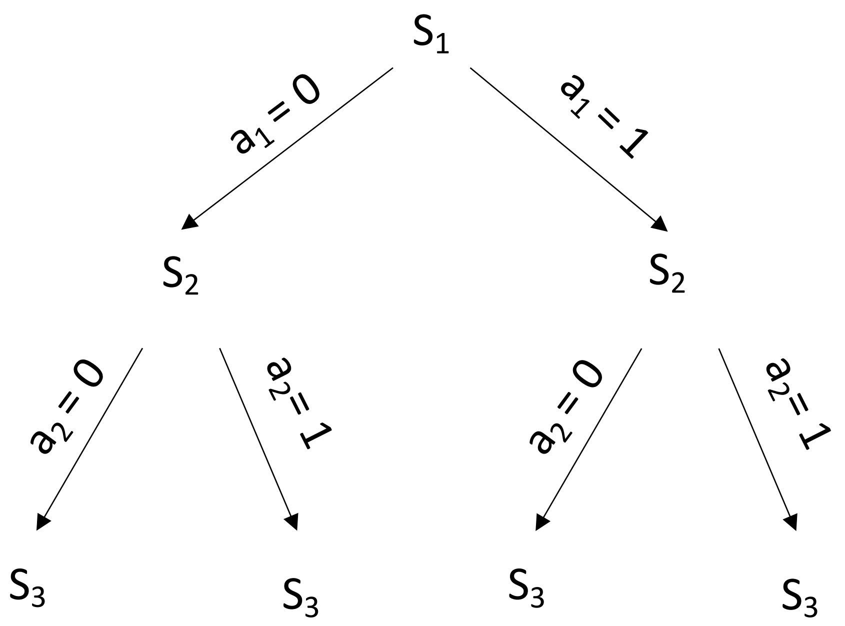
Also called the “Cross Entropy method”

Are we done?

- Pros:
  - Extremely simple
  - Very fast if parallelized
- Cons:
  - Does not work in high dimensional spaces (i.e., curse of dimensionality)

Can we do better?

- Monte Carlo tree search (MCTS)

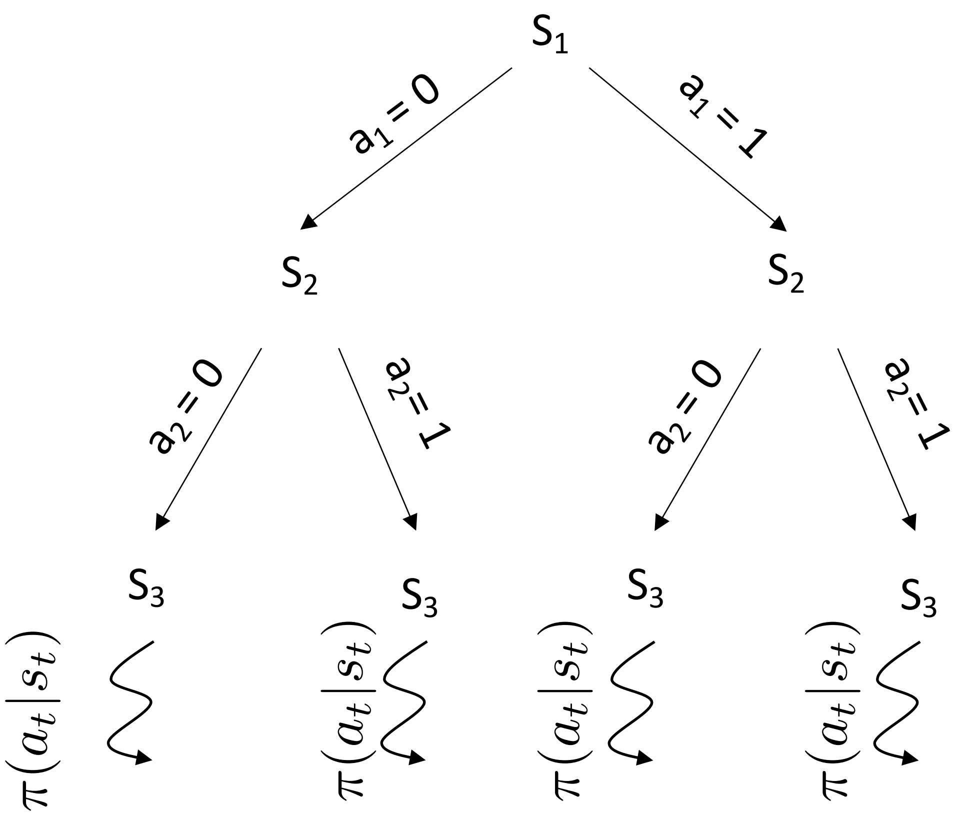


- Stochastic outcome, need to take each action multiple times
- Grows exponentially, not good enough

Monte Carlo tree search

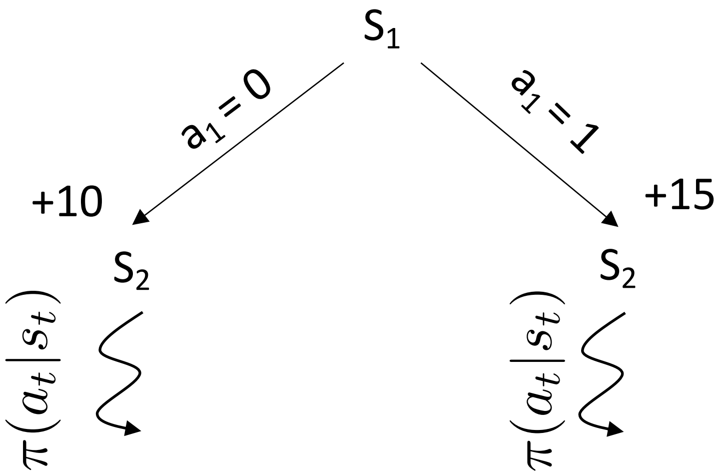
- Instead of expanding the tree all the way down, do it for a few steps and then run the policy

e.g., random policy



Monte Carlo tree search

- Can’t search all the paths (multiple times)
- Where to search first?



- We don’t want to give the same weight to every branch
- We want to choose nodes with the highest reward, but also explore from time to time

Monte Carlo tree search

Generic MCTS:

1. Find a leaf s_l using TreePolicy(s_1)
2. Evaluate the leaf using DefaultPolicy(s_l)
3. Update all values in the tree between s_1 and s_l

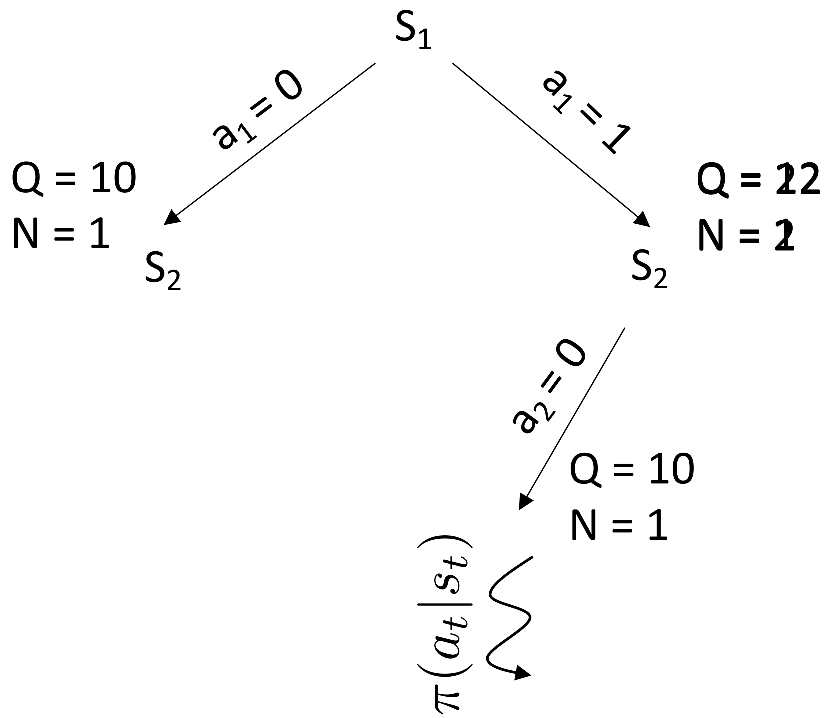
Take the best action from s_1

Algorithm 2 Greedy Tree policy

function TREEPOLICY(v)
  $v_{next} \leftarrow v$
  if $|Children(v_{next})| \neq 0$ **then**
    $a \leftarrow arg\max_{a \in A} Q(v, a)$
    $v_{next} \leftarrow nextState(v, a)$
    $v_{next} \leftarrow TreePolicy(v_{next})$
  return v_{next}

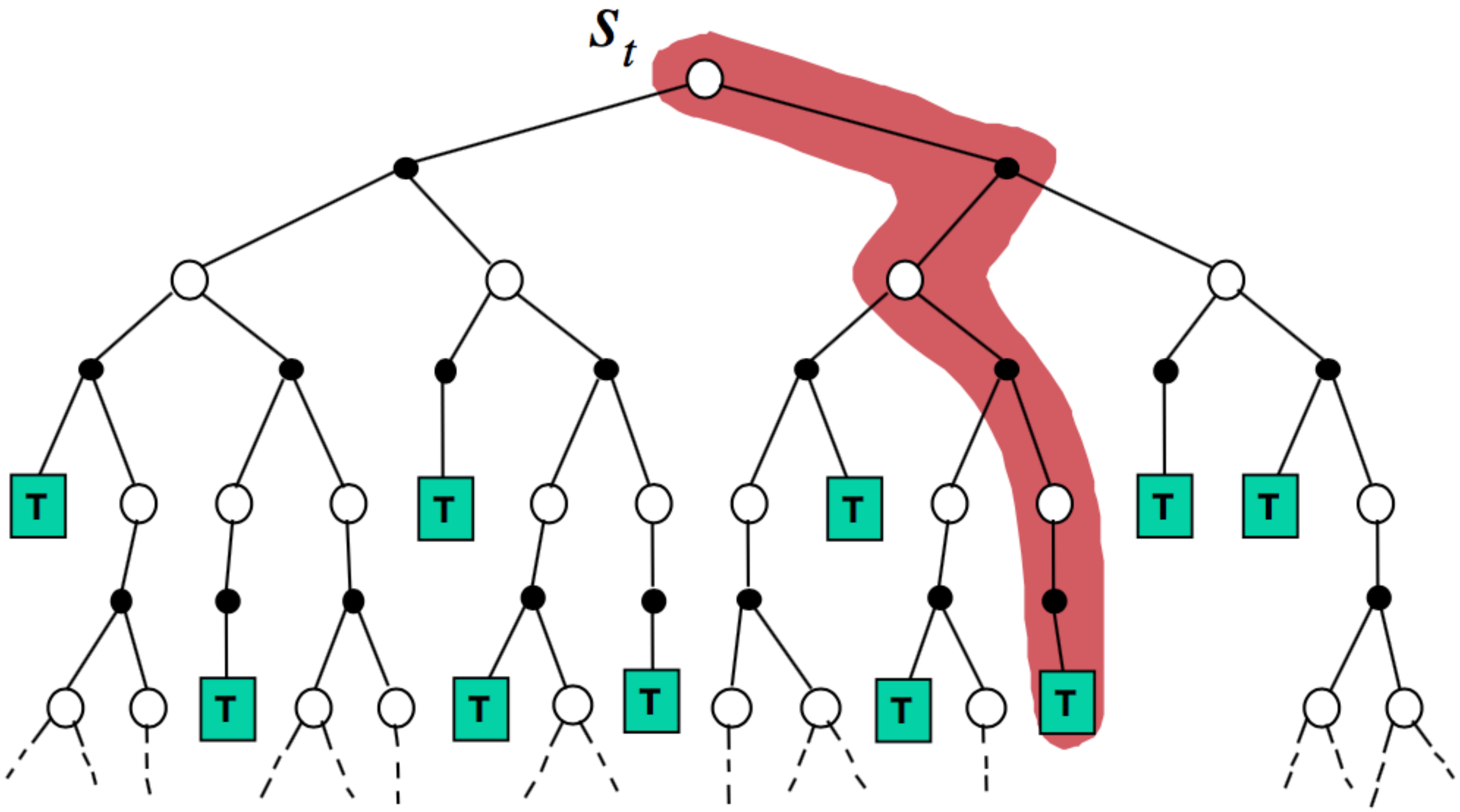
Algorithm 3 Upper Confidence Tree policy

function TREEPOLICY(v)
  $v_{next} \leftarrow v$
  if $|Children(v_{next})| \neq 0$ **then**
    $a \leftarrow arg\max_{a \in A} Q(v, a) + \sqrt{\frac{2 \log N(v)}{N(v,a)}}$
    $v_{next} \leftarrow nextState(v, a)$
    $v_{next} \leftarrow TreePolicy(v_{next})$
  return v_{next}



Credit: Sergey Levine

Monte Carlo tree search



Do we ever know the model?

- Sometimes we know the true model
  - Chess, Go
  - Trajectory of a (small, short-ranged) rocket
  - Simulated environments
    - We cannot write down the math, but we have access to the source code
- Sometimes we know the form of the model, but not the parameters
  - System Identification
- **Can we learn the model?**

Why learn the transition model?

- We can use the model for planning
  - Can do all the good things we talked about so far
- Fewer samples needed (from interaction with the real-world)

How can we learn the model?

1. Run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$
2. Learn model dynamics $p(s'|s, a)$ to minimize $\sum_i ||p(s'_i|s_i, a_i) - s'_i||$
3. Plan using the learned model $p(s'|s, a)$ to choose actions

What can go wrong?

- If our model is inaccurate, even small mistake will add up, leading to huge errors

What can we do?

- Re-plan often!

Credit: Sergey Levine

How can we learn the model?

1. Run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$
2. Learn model dynamics $p(s'|s, a)$ to minimize $\sum_i ||p(s'_i|s_i, a_i) - s'_i||$
3. Plan using the learned model $p(s'|s, a)$ to choose actions
4. Execute the first planned action, observe the next state
5. Append (s, a, s') to D

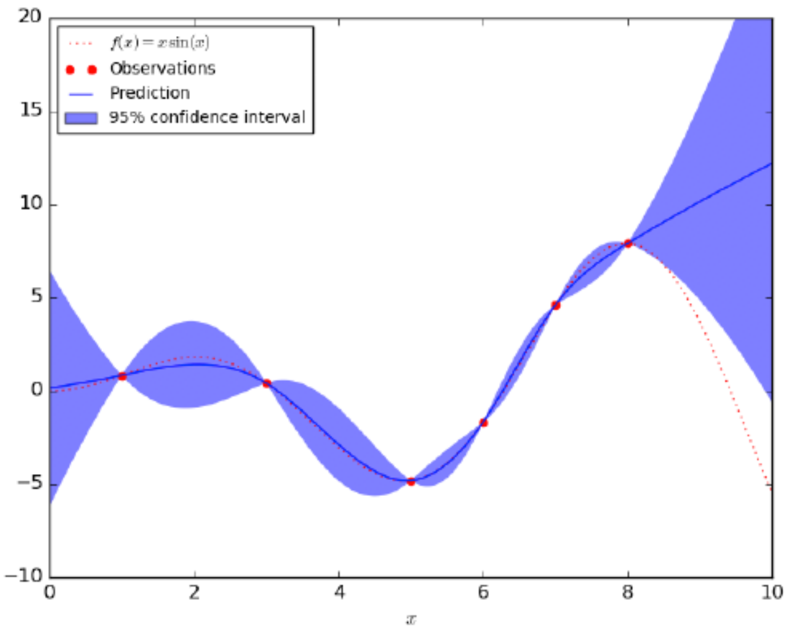
The more we re-plan, the less perfect our model needs to be

Credit: Sergey Levine

Incorporating uncertainty into decisions

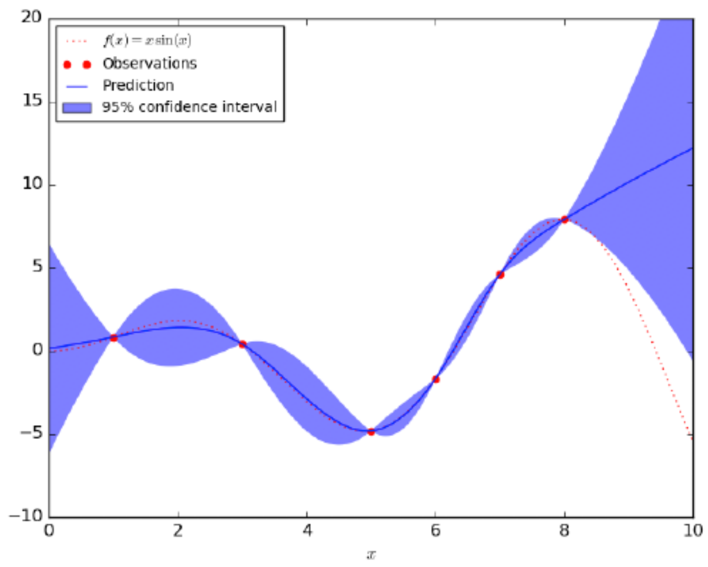
- In places where we have not seen a lot of data, our model has a higher uncertainty

- If we estimate the uncertainty associated with each prediction, then we can use the expected reward to avoid the high-variance states!

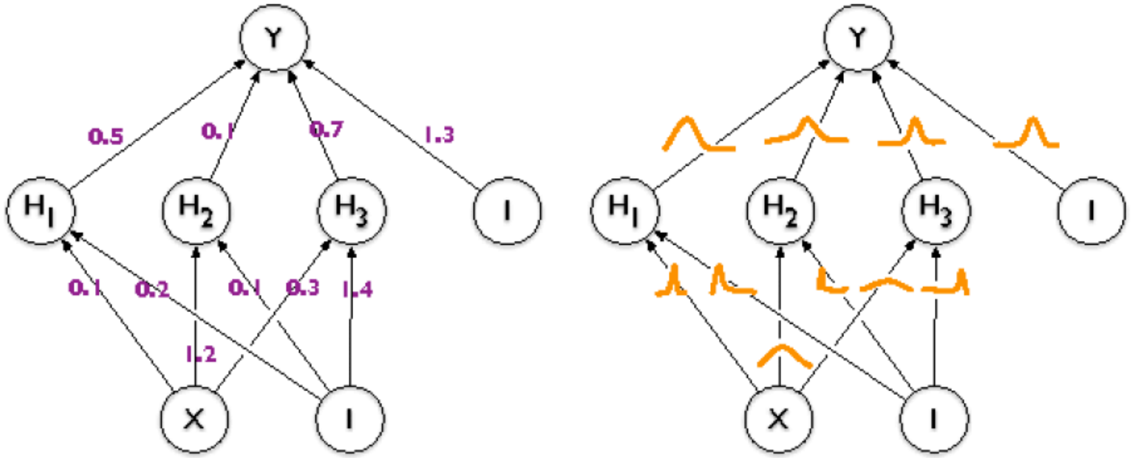


What are our options?

- Gaussian Processes



- Bayesian Neural Networks
  - Pyro, Edward2



Recap

- Value based
  - No policy (implicit)
  - Value function
- Policy based
  - Policy
  - No value function
- Actor critic
  - Policy
  - Value function

- Model-based
  - Transition and reward model
- Model-free
  - No transition and no reward model (implicit)