Reinforcement Learning – COMS4061A/7071A

Model-Based RL

Prof. Benjamin Rosman
Benjamin.Rosman1@wits.ac.za / benjros@gmail.com

Based heavily on slides by Steve James

Sutton and Barto [2018], Chapter 8

What is RL?

An approach that enables an agent to learn to act optimally through trial-and-error interaction with its environment.

But this could take forever!

Previously

Call this the "model"

- Dynamic programming:
  - Given transition dynamics and reward function
  - Compute value function/policy
- Model-free RL:
  - Learn a value function/policy from experience
- Model-based RL:
  - Learn a model from experience
  - Compute value function/policy

Learning and planning

- Learning: compute a model/value function/policy based on real-world experience
- Planning: compute a value function/policy based on simulated experience
- Question: What is dynamic programming?

What is dynamic programming?

An interesting question is, 'Where did the name, dynamic programming, come from?' The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical.

The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, 'programming.' I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying—I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

-Richard Bellman

Model-based RL

- General approach:
  - Collect data from the real world
  - Learn a model of the world from the data
  - Use that model to plan

Simulation
"in your head"

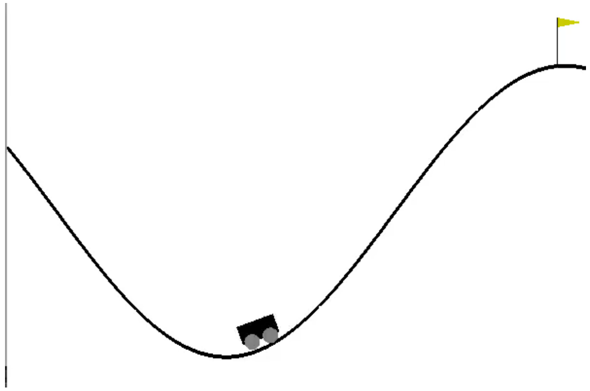
- Problems?
- Advantages?

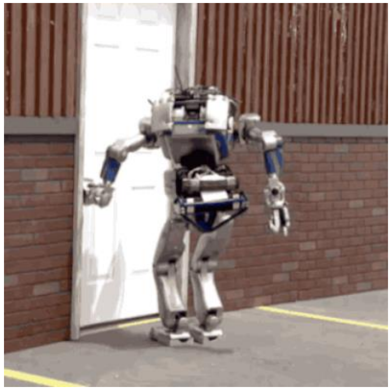
Sample efficiency

- Minimise our interaction with the world

```
env = gym.make('MountainCar-v0')
action = ...
env.step(action)
```

Minimise # step!





PILCO

Deisenroth, et al. *PILCO: A Model-Based and Data-Efficient Approach to Policy Search*

Models

- Model: anything the agent can use to predict how the environment will respond to its actions

1. Distribution model: description of all possibilities and their probabilities
   - e.g. $p(s', r \mid s, a)$ for all s, a, s', r
2. Sample model, a.k.a. a simulation model
   - produces sample experiences for given s, a
   - much easier to come by

- Both types of models can be used to produce hypothetical experience

Model learning

- Model is an estimate of a true MDP $\langle S, A, P, R, \gamma \rangle$
- Assume states and actions are known
- Then model $M_\eta = \langle \hat{P}, \hat{R} \rangle$ where $\hat{P}, \hat{R} \sim P, R$

- We can draw next state and reward samples from these

Model learning

- Estimate M_η from experience $\{S_1, A_1, R_2, \ldots, S_T\}$
- This is supervised learning!

$$S_1, A_1 \to R_2, S_2$$
$$S_2, A_2 \to R_3, S_3$$

Etc.

- Learning $s, a \to r$ is a regression problem
- Learning $s, a \to s'$ is a density estimation problem

Batch Table Lookup Model

- Count visits $N(s,a)$ to each state-action pair

$$\hat{P}(s'|s,a) = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t\ , S_{t+1} = s, a, s'\ )$$

$$\hat{R}(s,a) = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t =\ s, a) R_t$$

Online Table Lookup Model

- On receiving experience (s_t, a_t, r_t, s_{t+1}):

$$\hat{P}(s_{t+1}|s_t, a_t) = \hat{P}(s_{t+1}|s_t, a_t) + \alpha(1 - \hat{P}(s_{t+1}|s_t, a_t))$$

$$\hat{P}(\hat{s}|s_t, a_t) = \hat{P}(\hat{s}|s_t, a_t) + \alpha(0 - \hat{P}(\hat{s}|s_t, a_t))$$

$$\hat{R}(s_t, a_t) = \hat{R}(s_t, a_t) + \alpha\left(r - \hat{R}(s_t, a_t)\right)$$

Example

- Three states (A, B, C); 2 actions (p, q); 8 episodes

- A, p, 0, B, p, 0, C
- A, q, 0, B, q, 0, C
- B, p, 1, C
- B, p, 1, C
- B, q, 0, C
- B, p, 1, C
- A, p, 0, B, p, 0, C
- A, q, 0, B, p, 1, C

Example

- Three states (A, B, C); 2 actions (p, q); 8 episodes

- A, p, 0, B, p, 0, C
- A, q, 0, B, q, 0, C
- B, p, 1, C
- B, p, 1, C
- B, q, 0, C
- B, p, 1, C
- A, p, 0, B, p, 0, C
- A, q, 0, B, p, 1, C

State-action pair	Transition	Reward
(A, p)	B w.p. 1	0
(A, q)	B w.p. 1	0
(B, p)	C w.p. 1	1 w.p 2/3 0 w.p 1/3
(B, q)	C w.p. 1	0

Sample-based planning

- Generate samples from the model only!
  - Sample experience from model
- Apply model-free RL (MC/TD/etc) to samples

Random-sample one-step tabular Q-planning

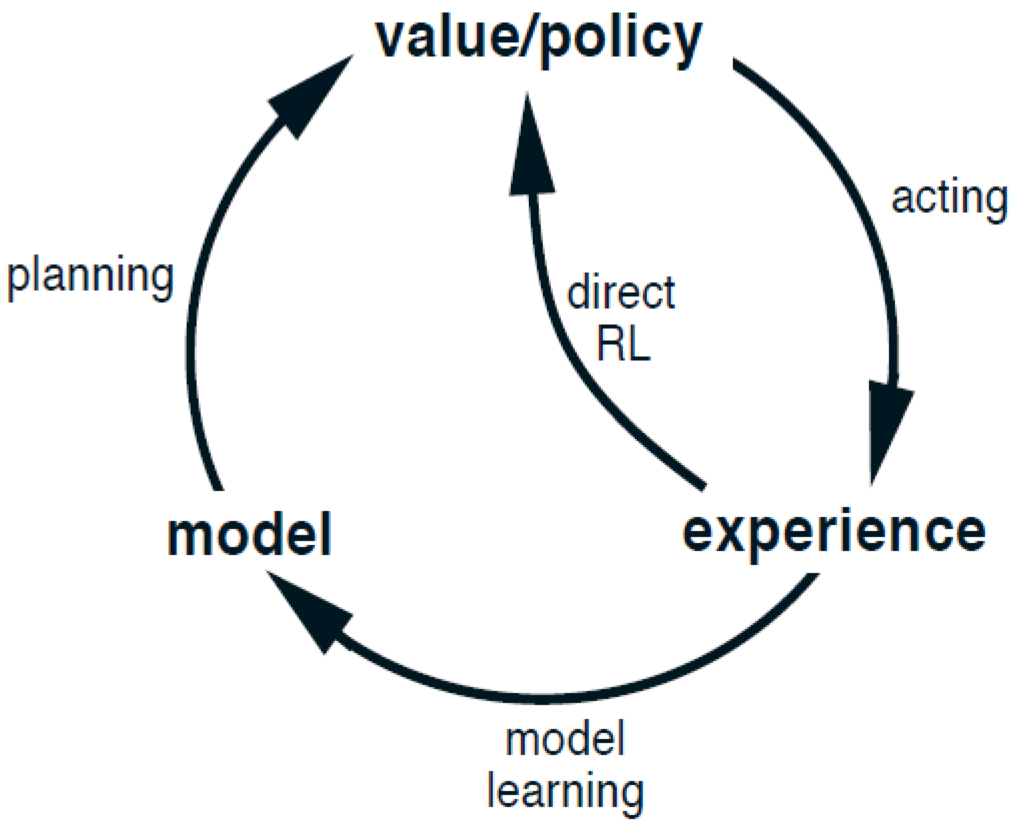
Loop forever:
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

Exact same Q-learning update!

Integrating planning and learning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
  - Learn a model from real experience
  - Plan value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn and plan value function (and/or policy) from real and simulated experience

Dyna



value/policy
acting
direct
RL
planning
model
experience
model
learning

Dyna

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S, Q)

(c) Take action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$ — Model-free Q-learning

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment) — Model learning

(f) Loop repeat n times: — Planning!

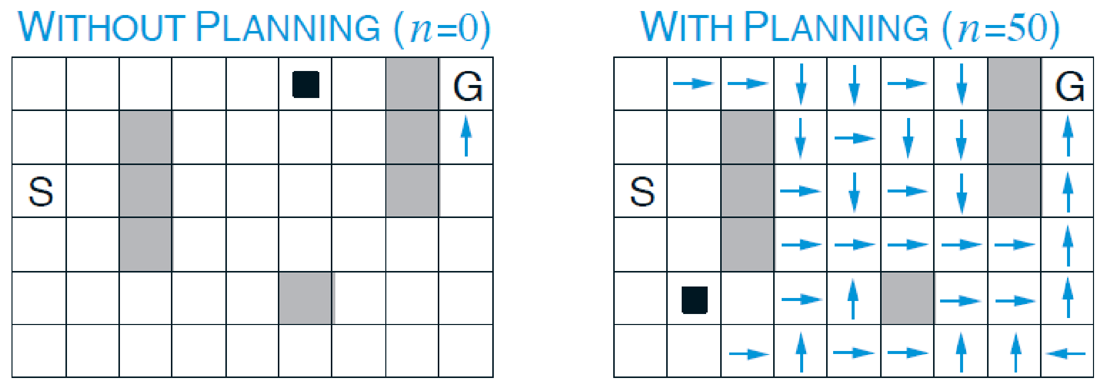
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

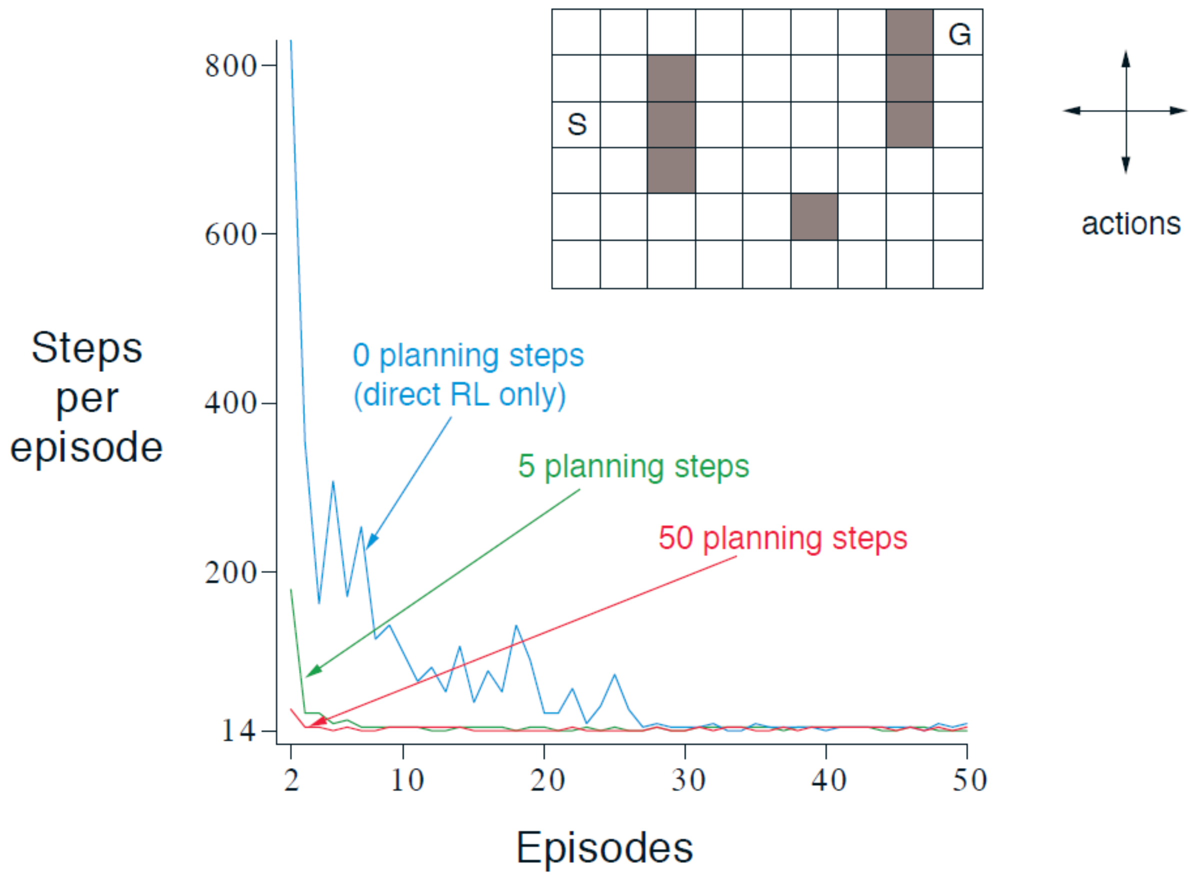
$R, S' \leftarrow Model(S,A)$

$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$ — Sample Q-planning

Dyna maze

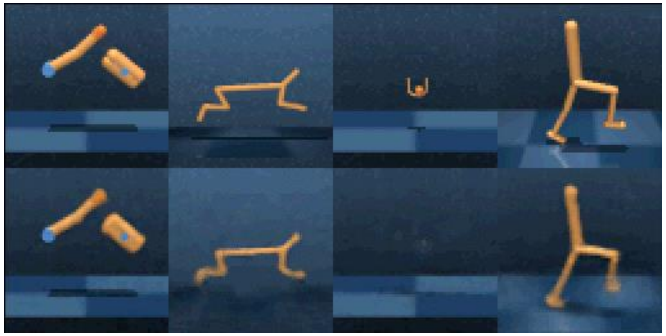


WITHOUT PLANNING (n=0)
WITH PLANNING (n=50)
G
S
G
S



G
S
actions
Steps per episode
800
600
400
200
14
0 planning steps (direct RL only)
5 planning steps
50 planning steps
2
10
20
30
40
50
Episodes

Model error



- What if the model is wrong?!
  - Planning would result in suboptimal policy
- Why might a model be wrong?
- In tabular case, simple heuristics can be effective

Stochasticity:

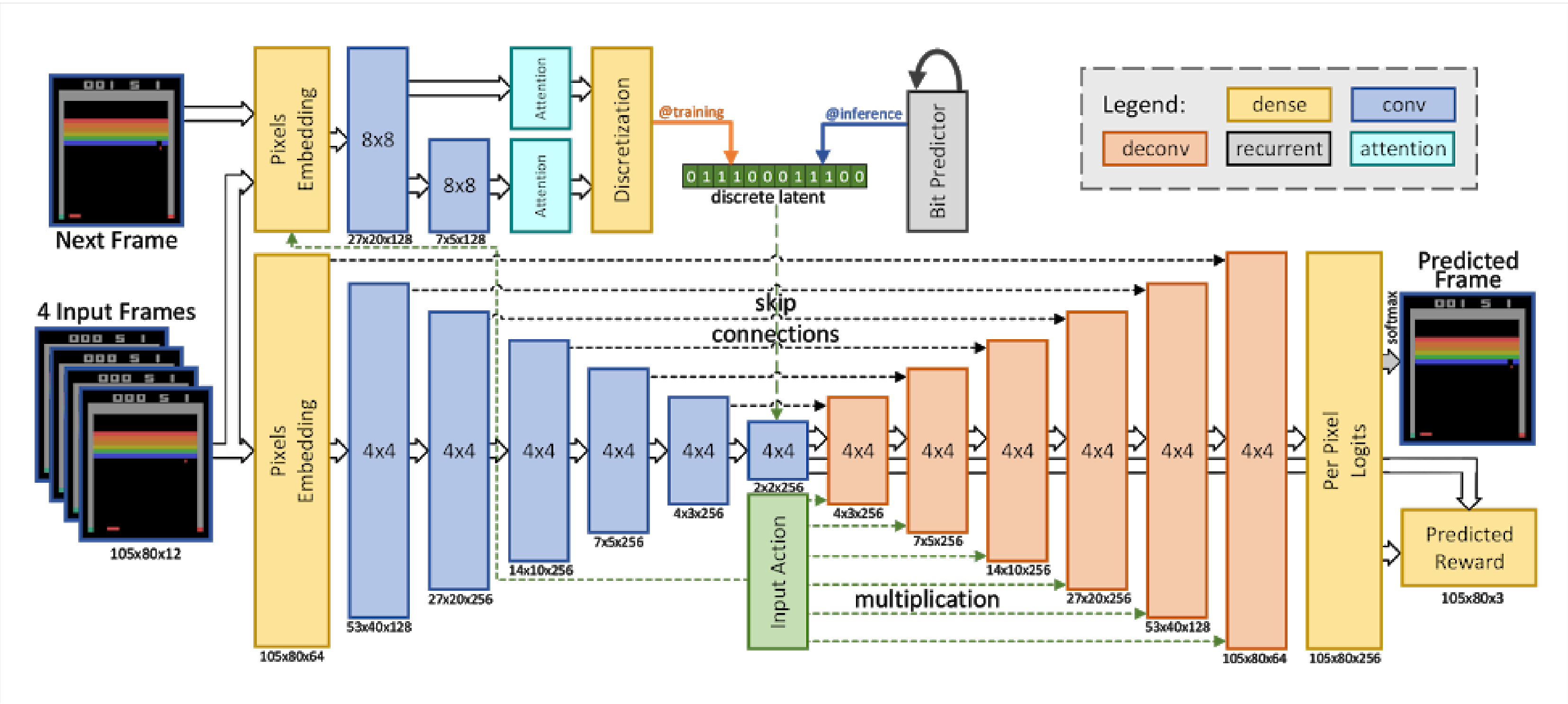
Dyna-Q+: give exploration reward bonus $\kappa\sqrt{\tau(s,a)}$

Time since last visit

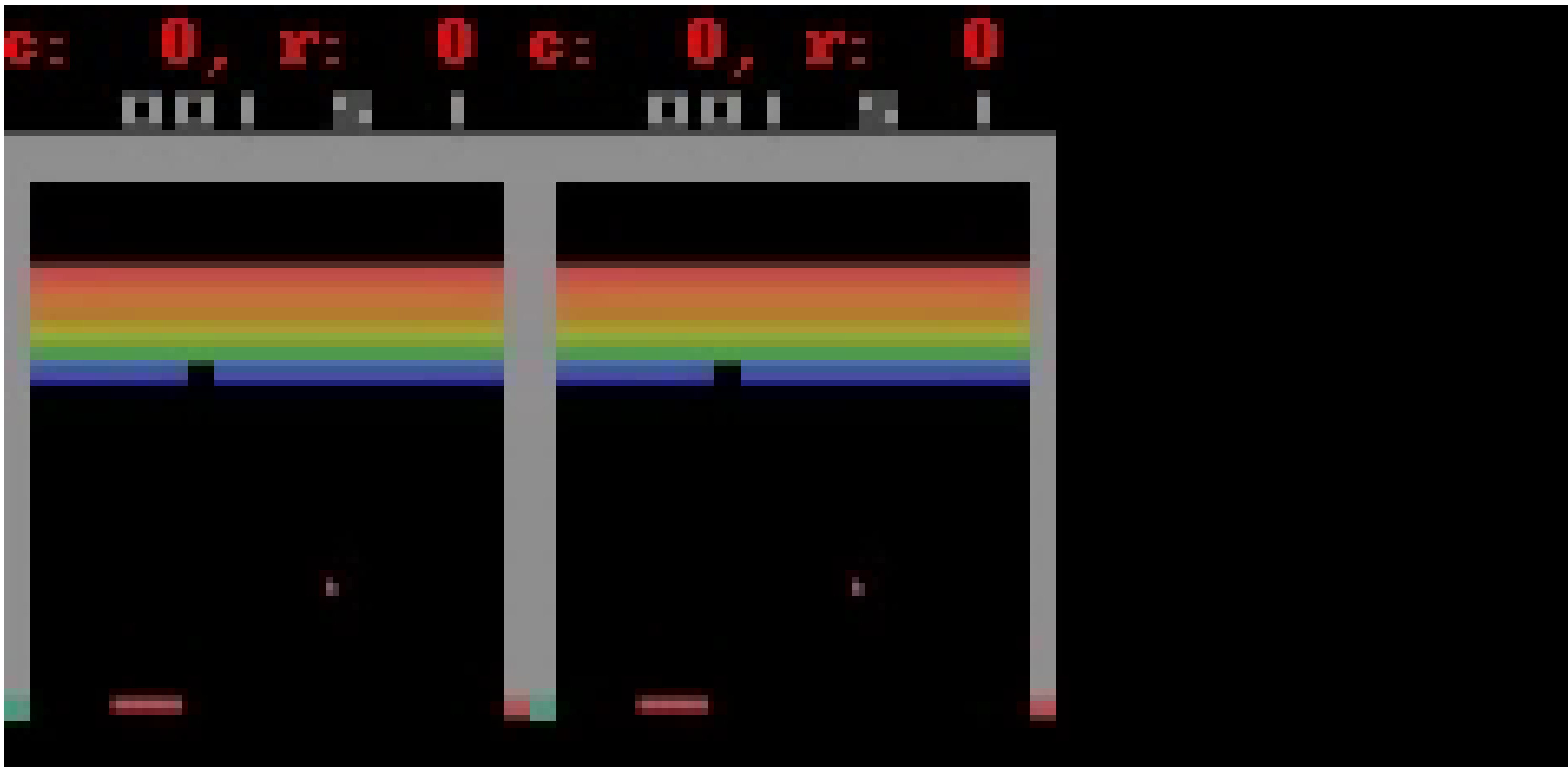
- Cutting-edge research for high-dimensional domains (e.g. PlaNet)

https://ai.googleblog.com/2019/02/introducing-planet-deep-planning.html

SimPle

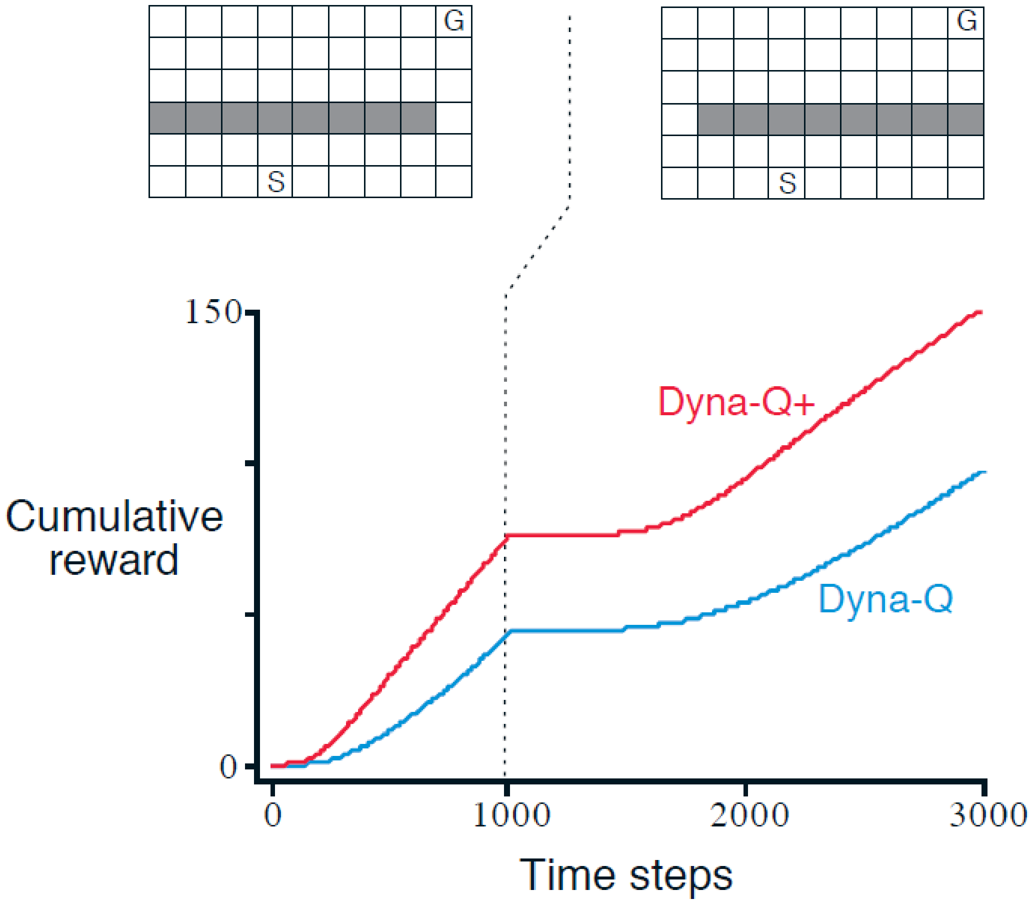


Next Frame
4 Input Frames
105x80x12
Pixels Embedding
105x80x64
8x8
27x20x128
8x8
7x5x128
Attention
Attention
Discretization
@training
@inference
0 1 1 1 0 0 0 1 1 1 0 0
discrete latent
Bit Predictor
Legend:
dense
conv
deconv
recurrent
attention
skip
connections
4x4
53x40x128
27x20x256
14x10x256
7x5x256
4x3x256
2x2x256
Input Action
multiplication
105x80x64
Per Pixel Logits
105x80x256
softmax
Predicted Frame
Predicted Reward
105x80x3

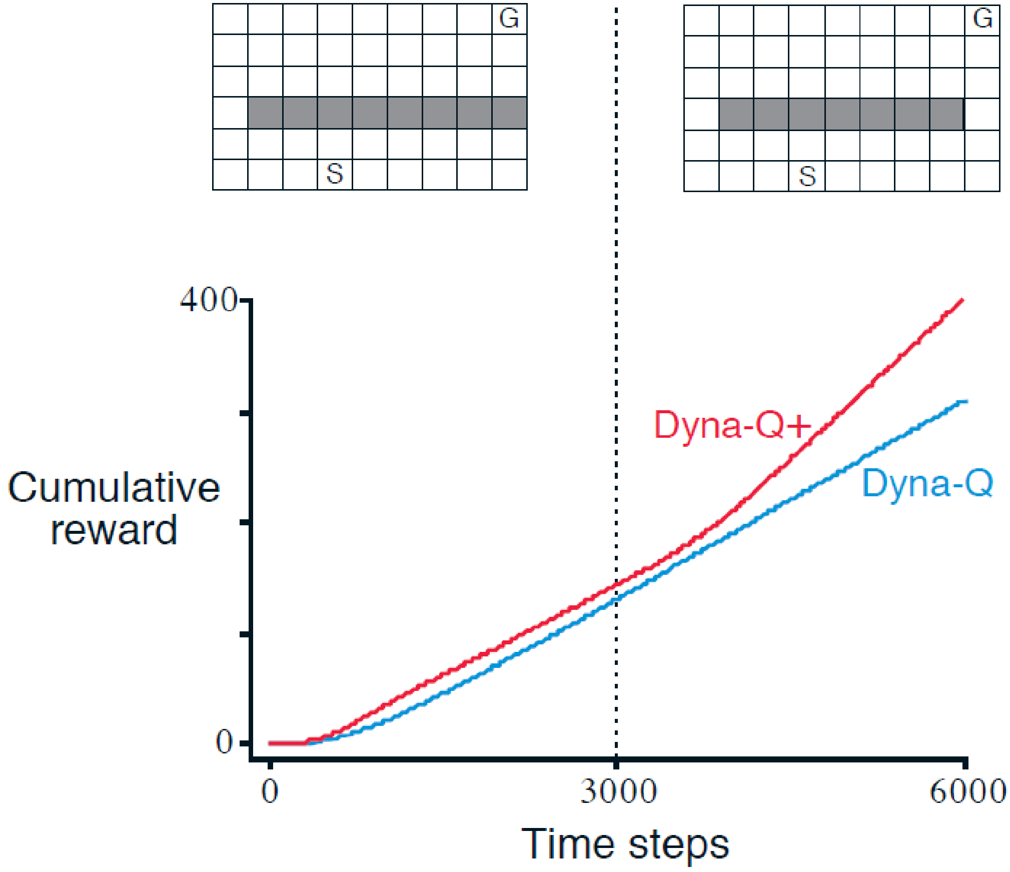

c: 0, r: 0
c: 0, r: 0


c: 3, r: 0
c: 7, r: 0

Environment changes



Environment changes



Back to Dyna

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow \varepsilon$-greedy(S,Q)
  (c) Take action A; observe resultant reward, R, and state, S'
  (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
  (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
  (f) Loop repeat n times:
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

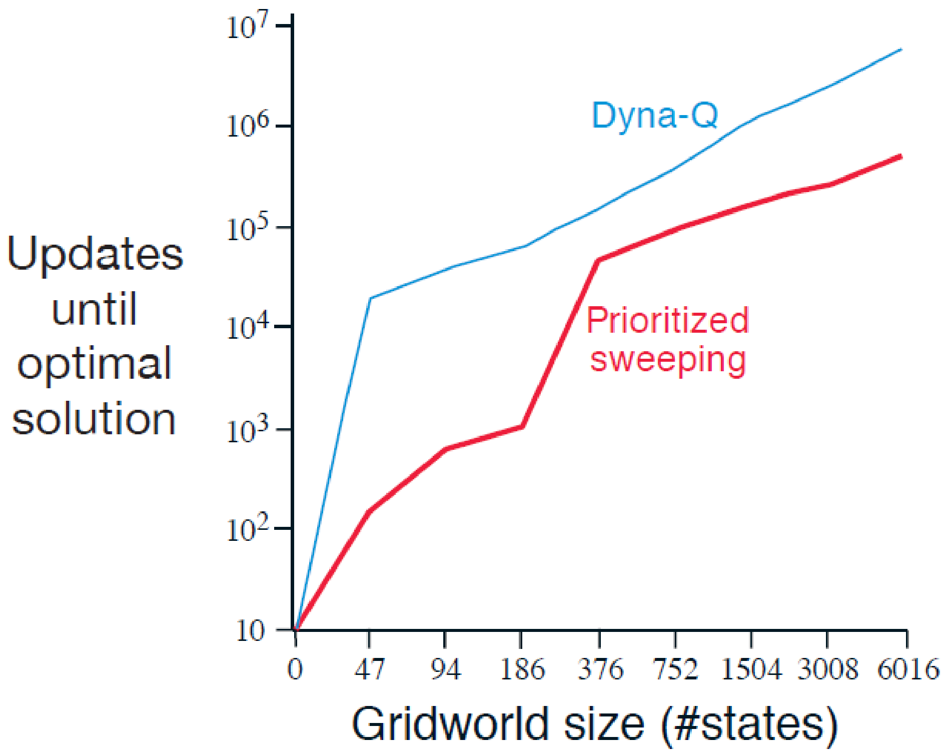
Can we do better than random?!

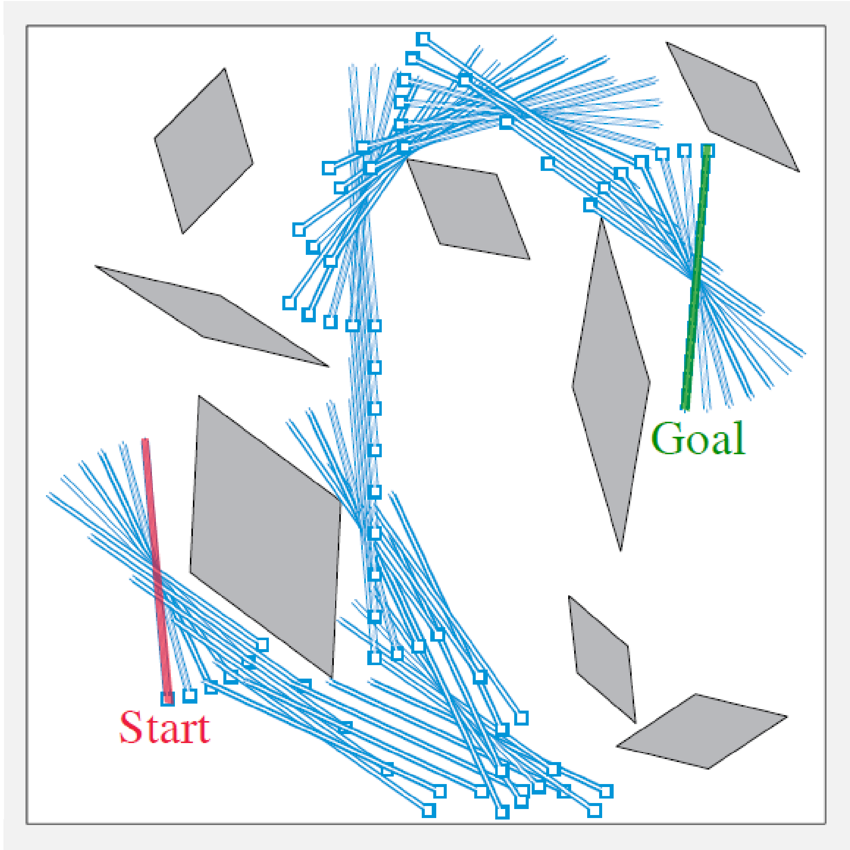
Prioritised sweeping

Prioritized sweeping for a deterministic environment

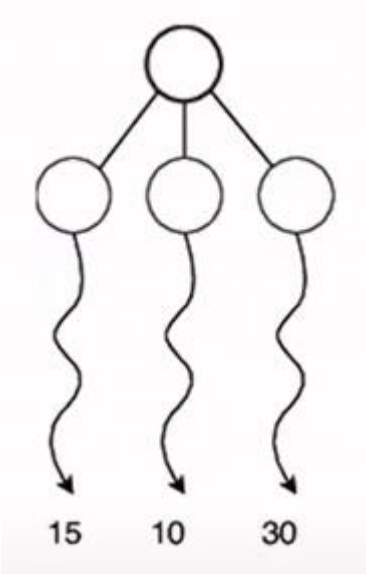
Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S, Q)$
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Model(S, A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$. How big a value change?
(f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
(g) Loop repeat n times, while $PQueue$ is not empty:
$S, A \leftarrow first(PQueue)$ Prioritise bigger changes!
$R, S' \leftarrow Model(S, A)$
$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
Loop for all $\bar{S}, \bar{A}$ predicted to lead to S: Update priorities for adjacent states
$\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
$P \leftarrow |\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})|$.
if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

Prioritised sweeping





Monte Carlo Search



- Using a model and a rollout policy π_d
  - **Simulate** many episodes with rollout policy
  - Compute mean return of episodes
  - This is $v_{\pi_{\mathrm{d}}}(s)$ or $q_{\pi_{\mathrm{d}}}(s, a)$

- Then act greedily
- By policy improvement theorem, this is better or equal to π_d
- Turns out to be quite effective!

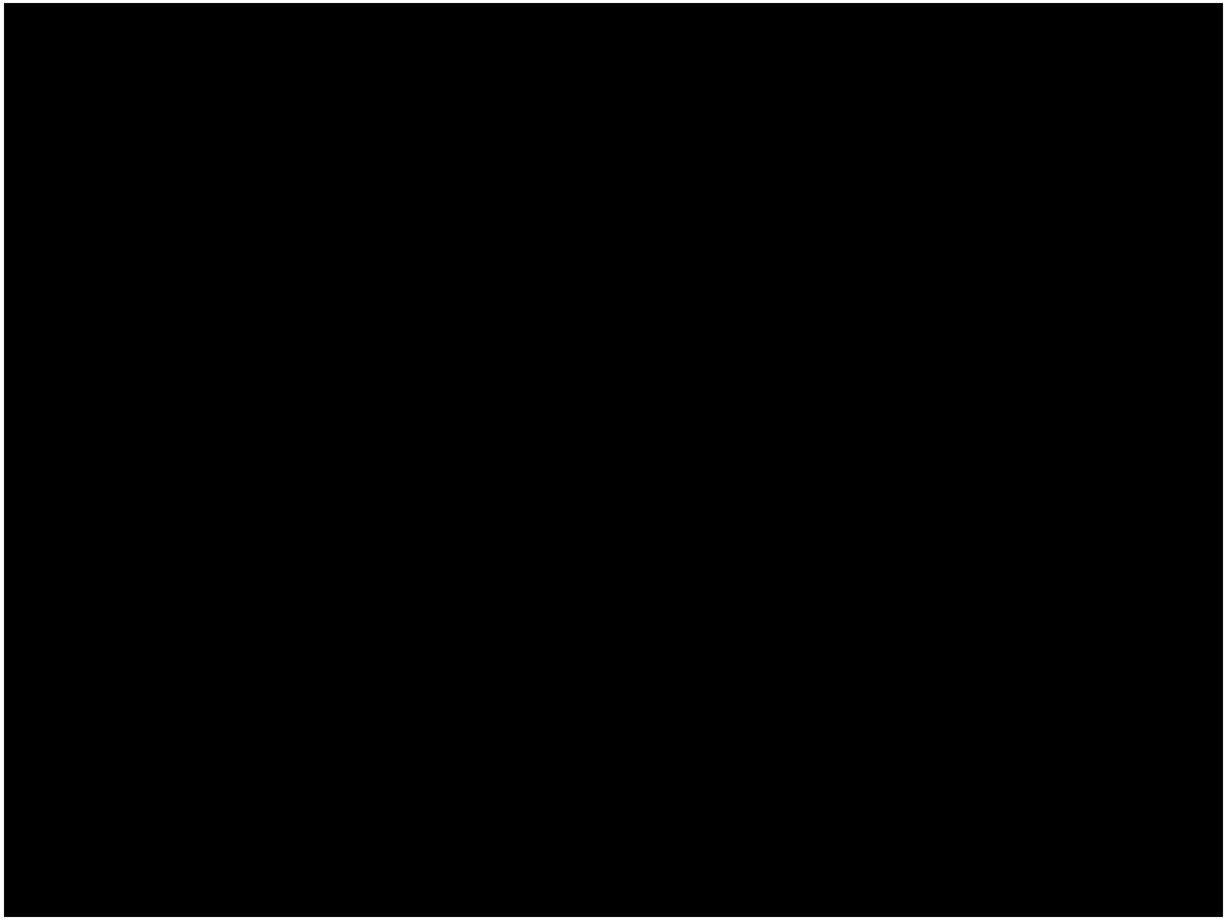
Monte Carlo Tree Search (MCTS)

- A key planning algorithm in RL
- Combine Monte Carlo search with bandit theory (remember UCB?)

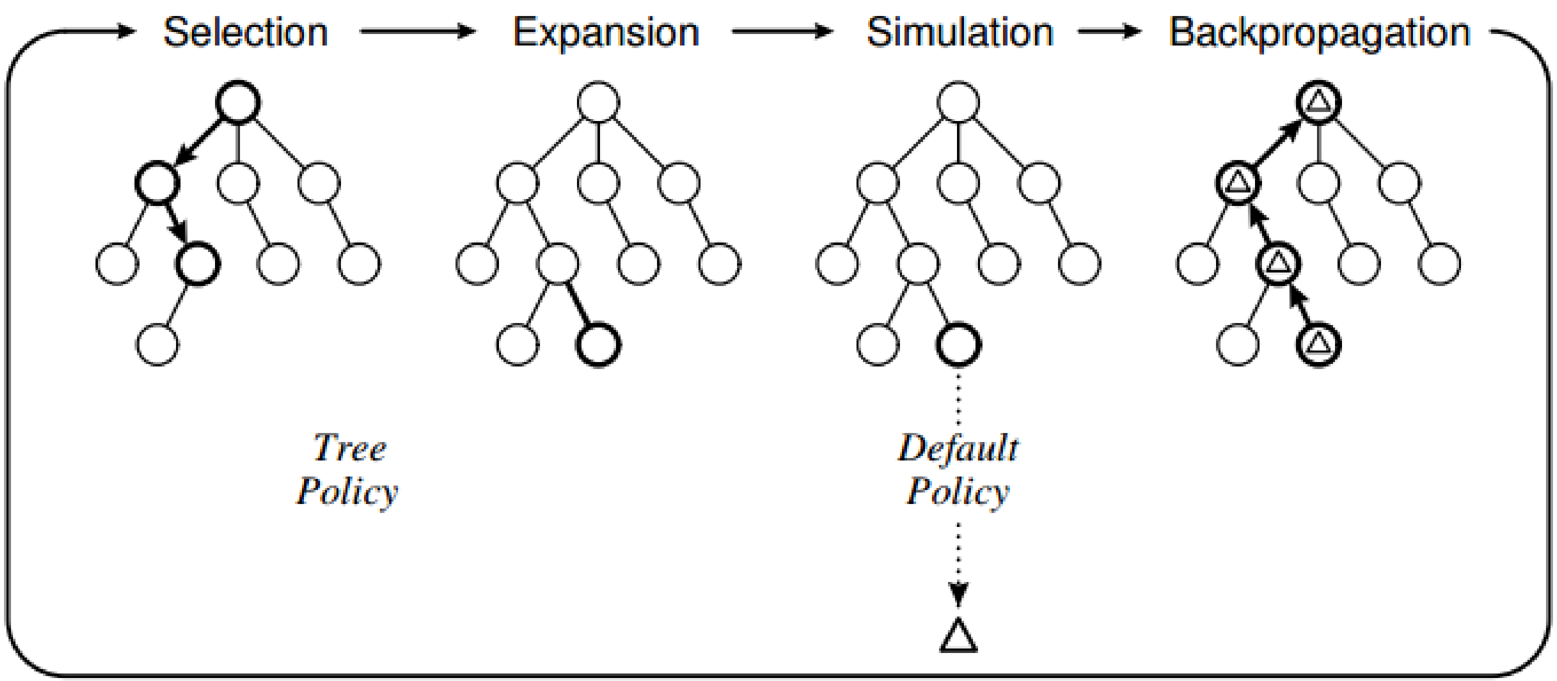
- At each state (node), store score and visit count
- Build an asymmetric search tree
- Visits most promising states more often
- Can stop planning and return action anytime

Coulom 2006, Kocsis & Szevespari 2007

MCTS



MCTS Phases



Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

Selection

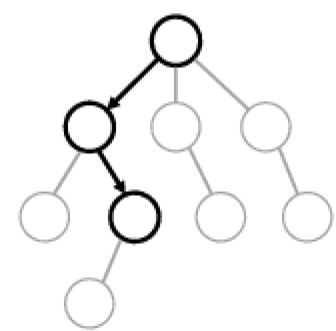
- Start at root node (current state)
- Use tree policy to descend tree to leaf node
- UCB1 selection: at each node s, select child i maximising

Visits of parent

Value of node

$$X_i + C_p \sqrt{\frac{\ln(n_s)}{n_i}}$$

Visits of node

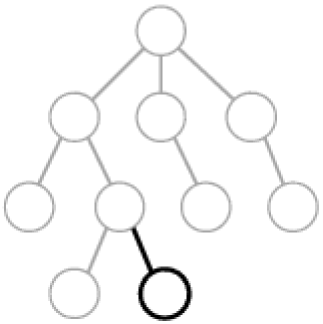


(a) Selection

- MCTS + UCB1 = UCT

Expansion

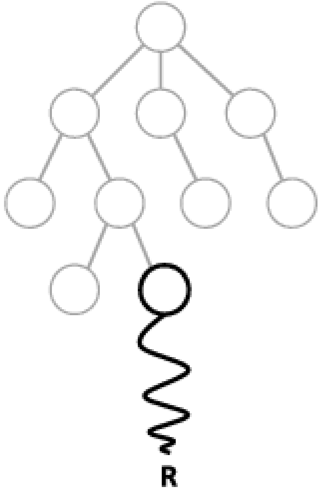
- At leaf node, take unexplored action
- Add new node to tree



(b) Expansion

Simulation

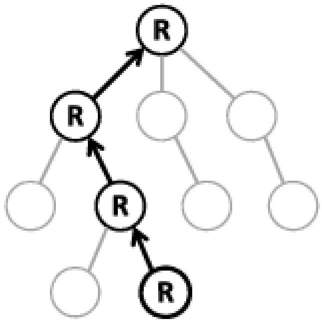
- From new node, use rollout policy to simulate trajectory
- At end of episode, get reward



(c) Simulation

Backpropagation

- Update all nodes from leaf to root with reward
- Update all nodes' visit counts by 1



(d) Backpropagation

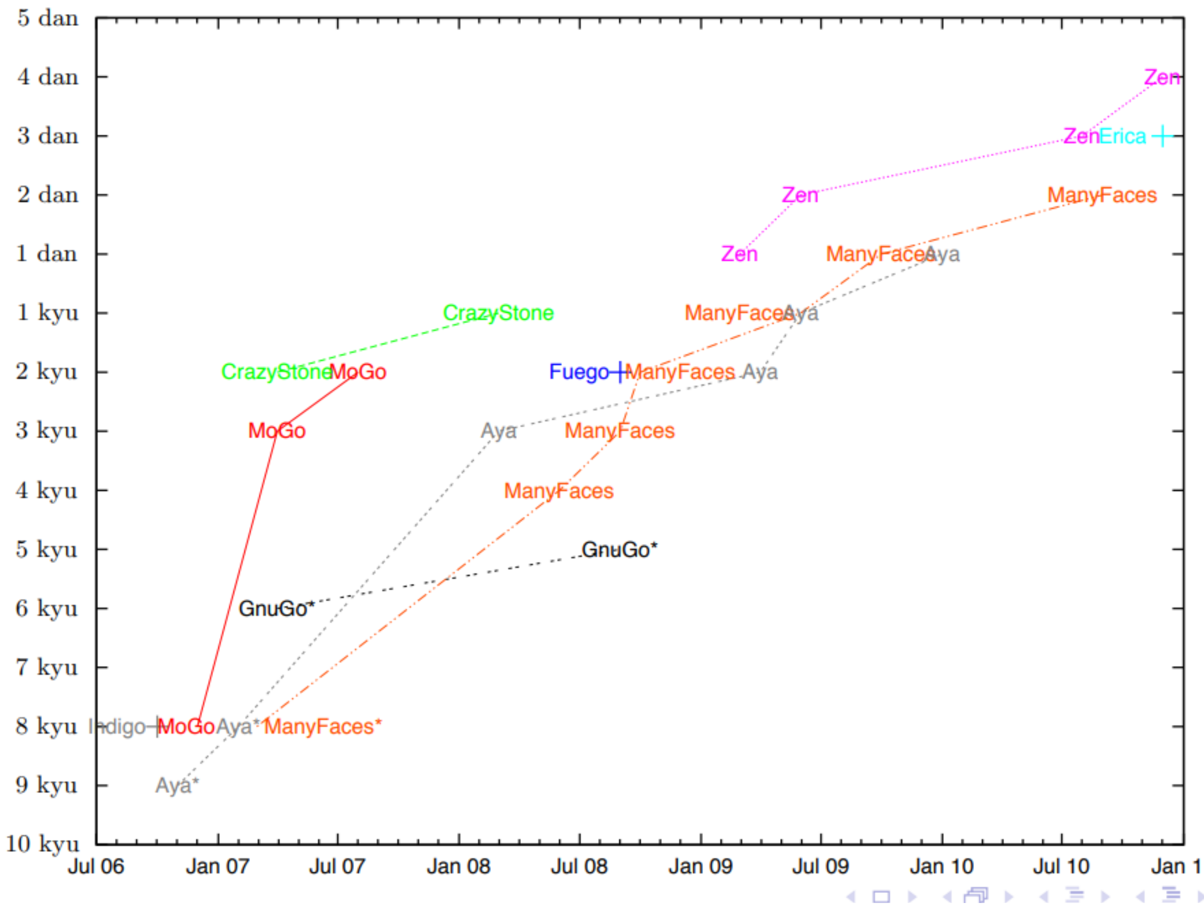
MCTS

- Continue cycle of selection, expansion, simulation and backprop until time runs out

- Select best action
  - Action that leads to state with highest value (max child)
  - Action that leads to state with most visits (robust child)

- Can reuse tree for next action!

MCTS for Go



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/dyna.pdf

Summary

- Distinction between learning and planning
- Use experience from real world to learn model
- Use model to update policy/value function
  - Improve sample efficiency
- Model error is an issue
  - Hard to learn models for complex environments
  - But *we* do it!

Homework

1. Think about your assignment
2. Implement your thoughts in the aforementioned step
3. Train your agent building on steps 1 and 2
4. Submit your assignment