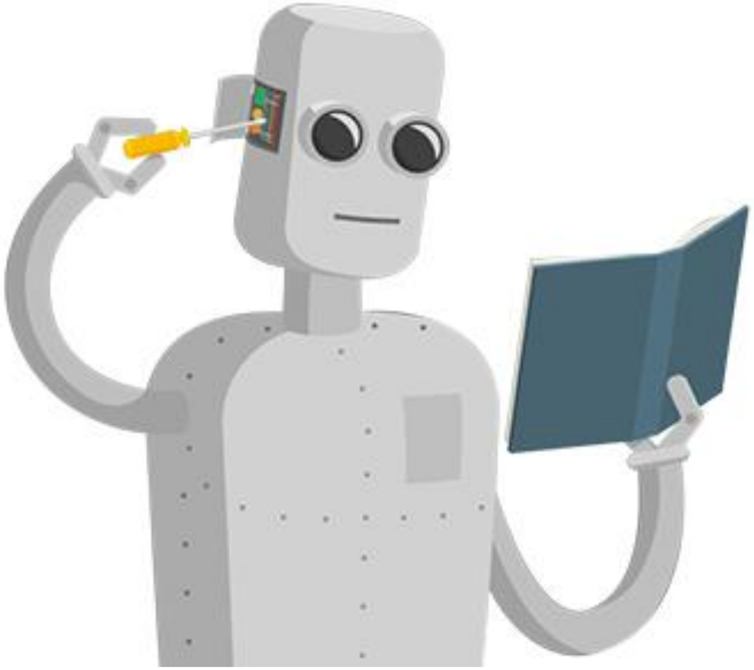
CSCE-642 Reinforcement Learning
Chapter 8: Planning and learning



Instructor: Guni Sharon

Solving MDPs so far…

Dynamic programming

- ✓ Off policy
- ✓ local learning, propagating values from neighbors
- ✗ assumes the model is known

Monte-Carlo

- ✗ On-policy (though important sampling can be used)
- ✗ Requires a full episode to train on
- ✓ Model free

TD Learning

- ✓ Off policy
- ✓ local learning, propagating values from neighbors
- ✗ Slow value propagation
- ✓ Model free

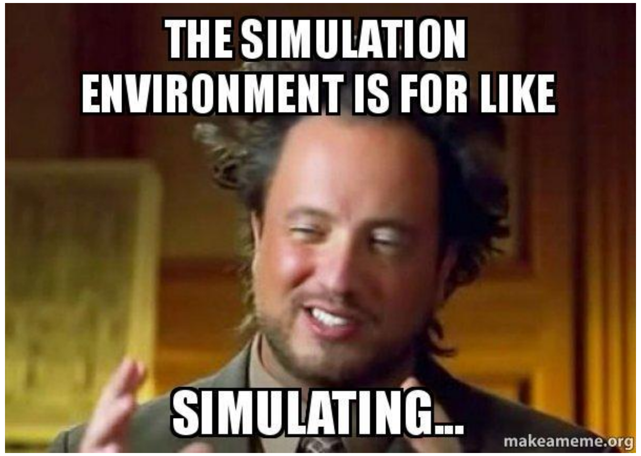
n-step TD Learning

- ✗ On-policy (though important sampling can be used)
- ✓ local learning (flexible locality)
- ✓ Model free

Must online solvers be Model free?

The value of a model

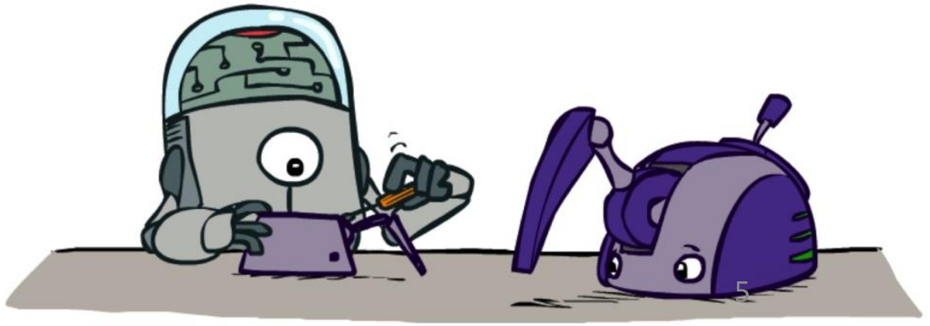
- Having a model is useful
  - We can simulate episodes cheaply
  - Compute optimal policy offline (no growing pains)
- Sounds great! We should use Model-based approaches
- For most practical applications, the model is unknown
- Can we learn the model (transition and reward functions)?
  - Sure, we just need to observe enough relevant transitions



Build a model

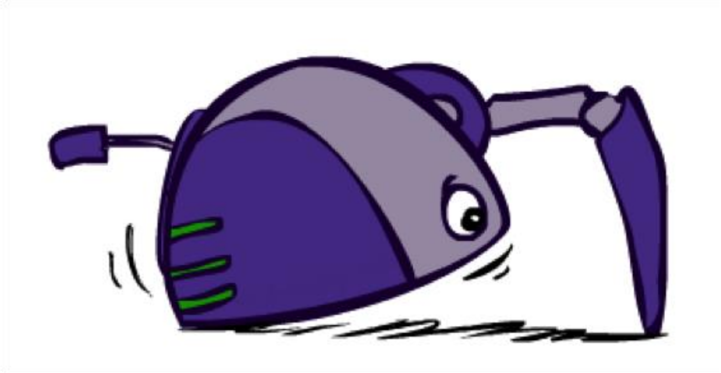
Unbiased estimations

- Observe transitions of type:
  - $< S_i, A_i, R_{i+1}, S_{i+1} >$
- Learn the transition probs: $P(s'|s,a) = \frac{N(s,a,s')}{N(s,a)}$
  - Where $N(x, y, z)$ is the number of observations that include events x **and** y **and** z
- Learn a reward function: $R(s,a) = \text{AVG}\big(r(s,a)\big)$
- Now solve offline!
  - What's the problem(s)?



Learning a model

- ☑ A model allows offline planning
- ✕ Learning a model is done online
- How many samples are required to learn an accurate model?
  - Can be arbitrarily large
- How many samples are required to learn a useful model?
  - Depends on the model and reward distribution
  - E.g., what’s the expected reward from buying a Powerball ticket
- Eventually, a model is just a model

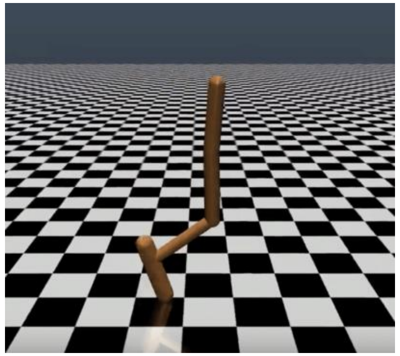


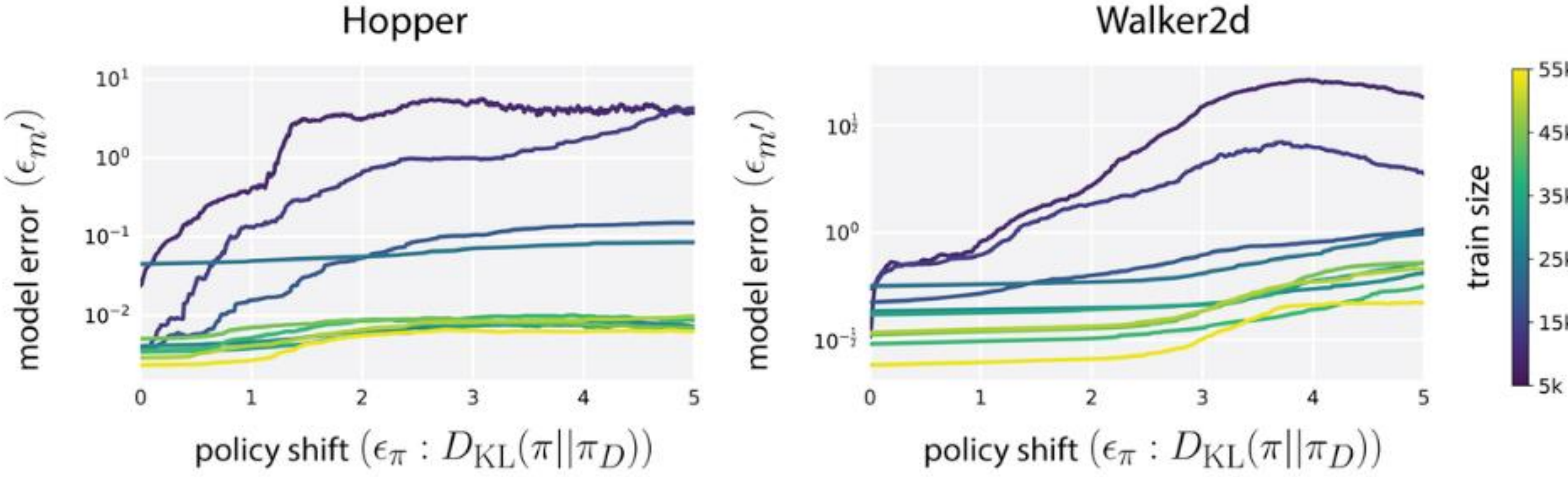
Can we train models for free?

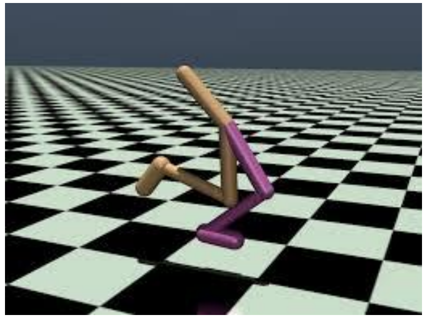
- Training a model might require many transition observations
  - $< S_i, A_i, R_{i+1}, S_{i+1} >$
- Any online learning method (e.g., Q-learning) generates such observations
- Let’s use these observations to learn a model
  - Its free!

Off-policy learning with approx. model

- Generalization of learned models, trained on samples from a data-collecting policy π_D, to the state distributions of a target policy π
- Increasing the training set size not only improves performance on the training distribution, but also on nearby distributions







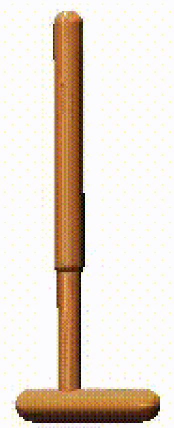
Results by Sergey Levin

Model approx. looks promising

- Not so fast…
- The results suggest that a single-step predictive accuracy of a learned model can be reliable under policy shift
- The catch is that most model-based algorithms rely on models for much more than a single-step
- When predictions are strung together, small errors compound over the prediction horizon

Error accumulation

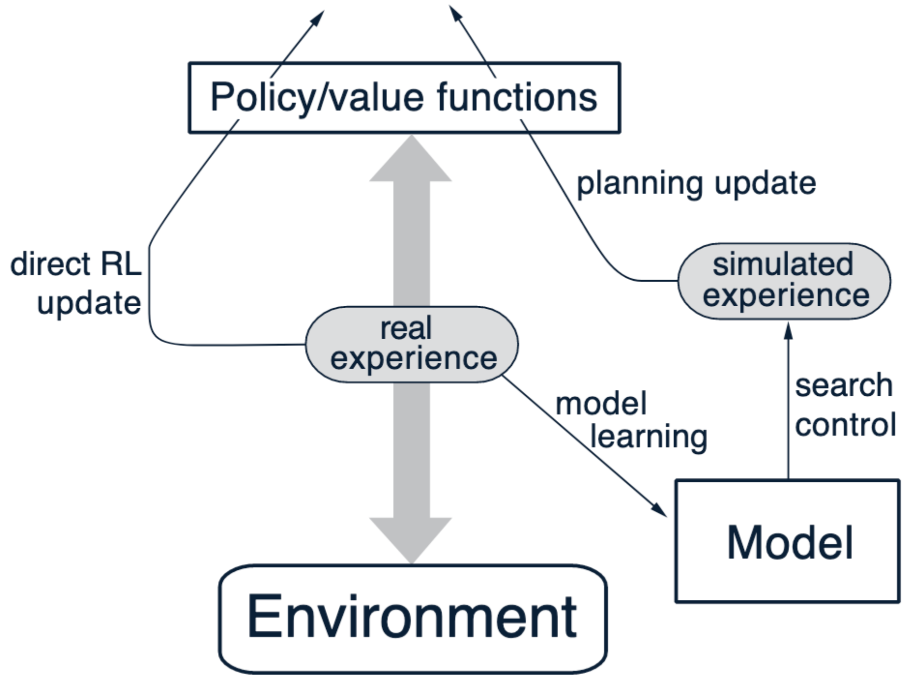
- A 450-step action sequence rolled out under a learned probabilistic model
- Depicting the mean prediction and one standard deviation away from the mean colored regions
- The growing uncertainty and deterioration of a recognizable sinusoidal motion underscore accumulation of model errors



Demo by Sergey Levin

Experience from simulation

- Observations from the real world might be costly and dangerous
- If we have a model, we can generate simulated observations
- Simulated observations are faster, cheaper, and safer to obtain
- Use the simulated experience to train the policy with any online method
- Be careful: Modeling errors can cause diverging TD updates



Simulated trajectory (rollout) length

- There is a tension involving the model rollout length
- The model serves to reduce off-policy error over entire episodes
- However, increasing the rollout length also brings about increased discrepancy proportional to the model error
- Predictive models can generalize well enough for the incurred model bias to be worth the reduction in off-policy error, but
- Compounding errors make long-horizon model rollouts unreliable

So, what should we do?

- We observe that a one step rollout is mostly accurate
- A simple recipe is to use the model only to perform short rollouts from previously encountered (real) states instead of full-length rollouts from the initial state distribution
- This is the main idea behind the Dyna-Q algorithm

Dyna-Q

Tabular Dyna-Q

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

Do forever:

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

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

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

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

Q-learning

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

(f) Repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

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

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

Dyna-Q

Tabular Dyna-Q

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

Do forever:

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

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

(c) Execute 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) } Update model

(f) Repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

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

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

Dyna-Q

Tabular Dyna-Q

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

Do forever:

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

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

(c) Execute 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) Repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

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

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

Simulate transitions
and perform Q-learning

Dyna-Q

Tabular Dyna-Q

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

Why simulate only previously observed (S,A)?

*Else we don't have a reliable approximation

Dyna-Q

Tabular Dyna-Q

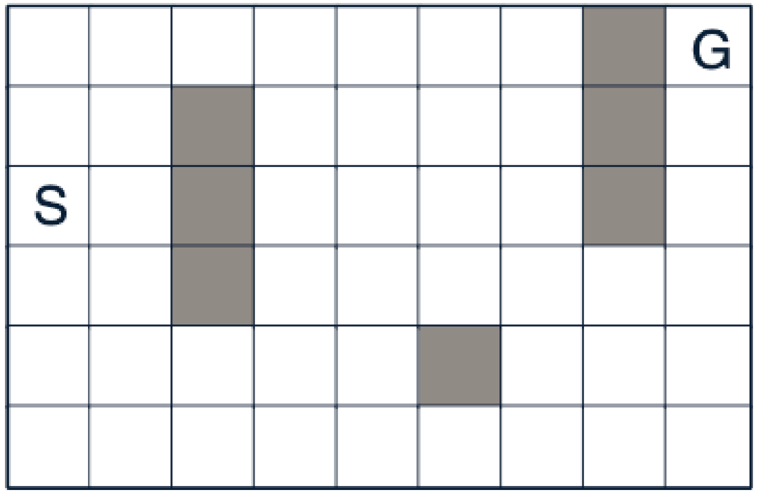
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \epsilon$-greedy(S,Q)
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R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

How should we set n?

*take into account model accuracy and cost of real vs simulated transition

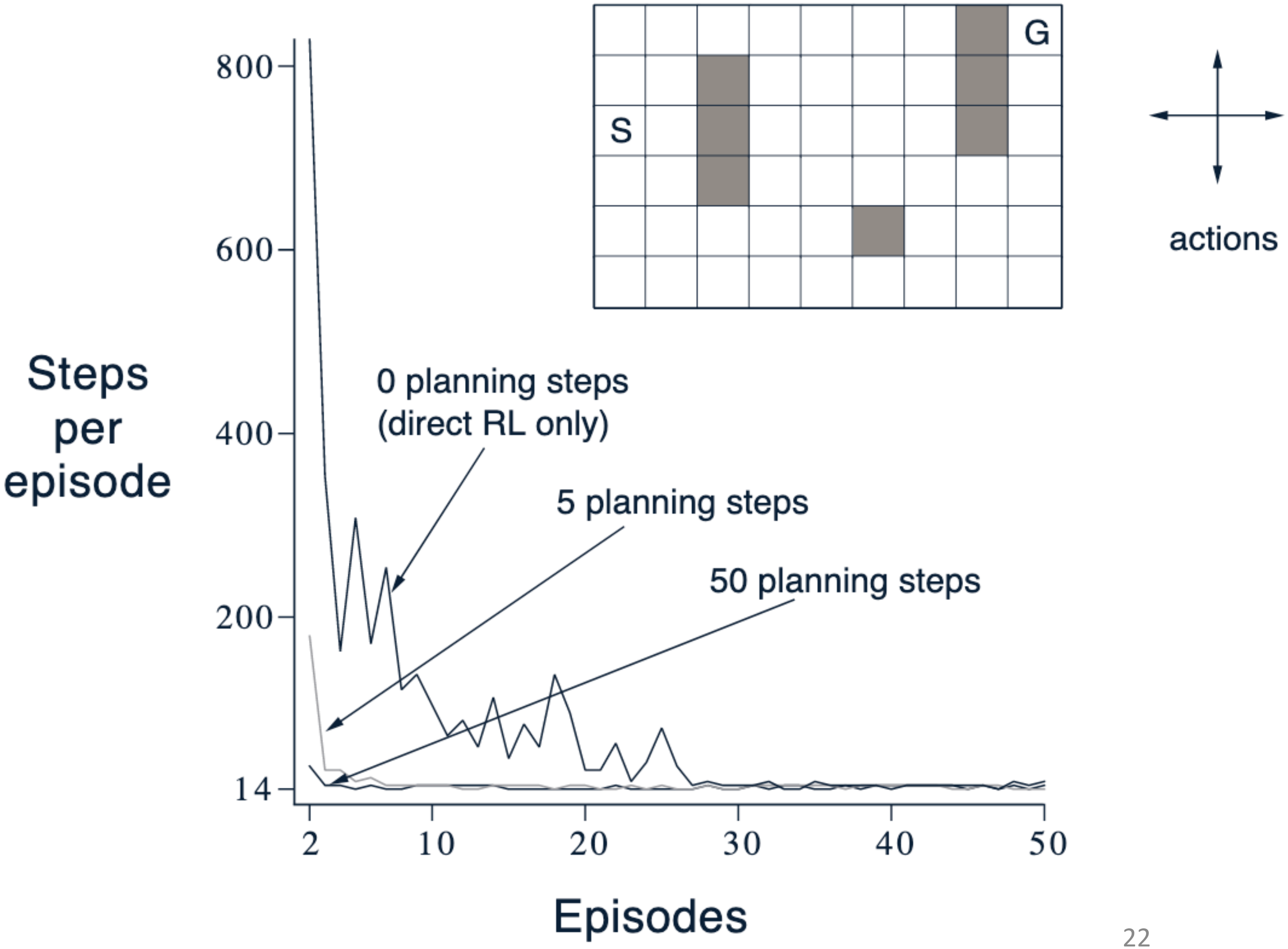
Dyna-Q example

- Consider a 4-connected grid
- Deterministic action outcomes (N,E,S,W)
- +1 reward for moving into G
- +0 otherwise
- $\gamma = 0.95$
- For Dyna-Q:
  - $\alpha = 0.1$
  - $\varepsilon = 0.1$
  - $n =$ varying

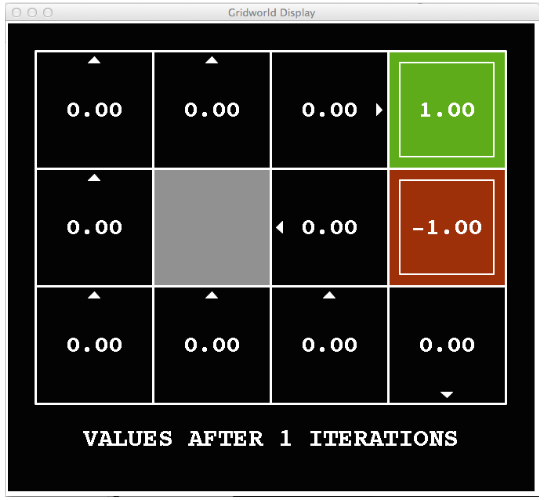


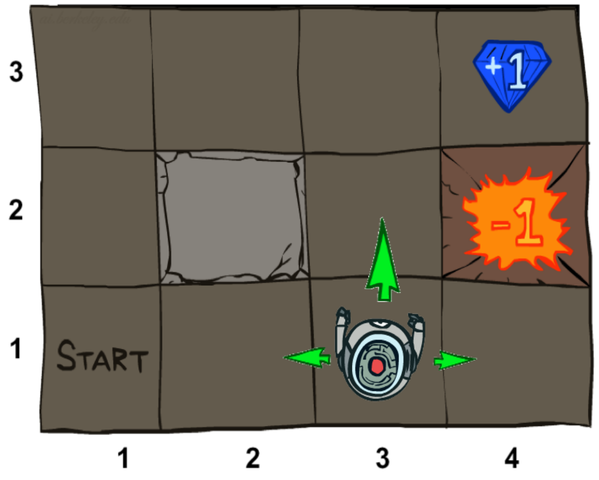
Dyna-Q example

- How accurate is the learned model?
- What would happen if the model wasn’t accurate?
- What’s more common in stochastic environments?
- What can we do about it?
  - Constantly explore!
  - See Dyna-Q+
    - (Ch. 8.3 in the textbook)



Simulating what matters





Tabular Dyna-Q

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

Do forever:

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

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

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

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

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

(f) Repeat n times:

$\quad S \leftarrow$ random previously observed state

$\quad A \leftarrow$ random action previously taken in S

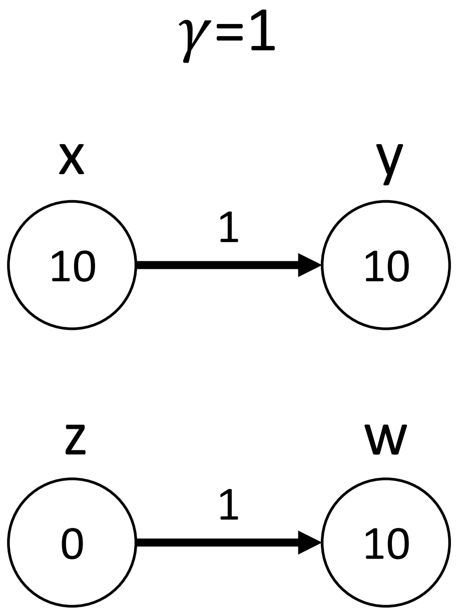
$\quad R, S' \leftarrow Model(S,A)$

$\quad Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

Do some (S,A) pairs matter more than others?

Prioritizing experience

- Which should have a higher priority?
  - $(x, \rightarrow)$
  - $(z, \rightarrow)$
- Larger updates (|TD error|) -> higher priority
- Prioritize updates according to TD error (δ)
- $\delta(S, A) = R + \gamma \max_a Q(S', a) - Q(S, A)$
- $Priority(S, A) = |\delta(S, A)|$
- $Priority(x, \rightarrow) =$
- $Priority(z, \rightarrow) =$



Prioritized sweeping

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Do forever:

(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S,Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Model(S,A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
(f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
(g) Repeat n times, while $PQueue$ is not empty:
  $S, A \leftarrow first(PQueue)$
  $R, S' \leftarrow Model(S,A)$
  $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
  Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:
    $\bar{R} \leftarrow$ 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

Store each observed transition in reply buffer with the affiliated priority

Prioritized sweeping

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty

Do forever:

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

(b) $A \leftarrow policy(S, Q)$

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

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

(e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.

(f) if $P > \theta$, then insert S, A into $PQueue$ with priority P

(g) Repeat n times, while $PQueue$ is not empty:

$S, A \leftarrow first(PQueue)$

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

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

Get max priority transition and update the relevant Q value

Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:

$\bar{R} \leftarrow$ 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

Prioritized sweeping

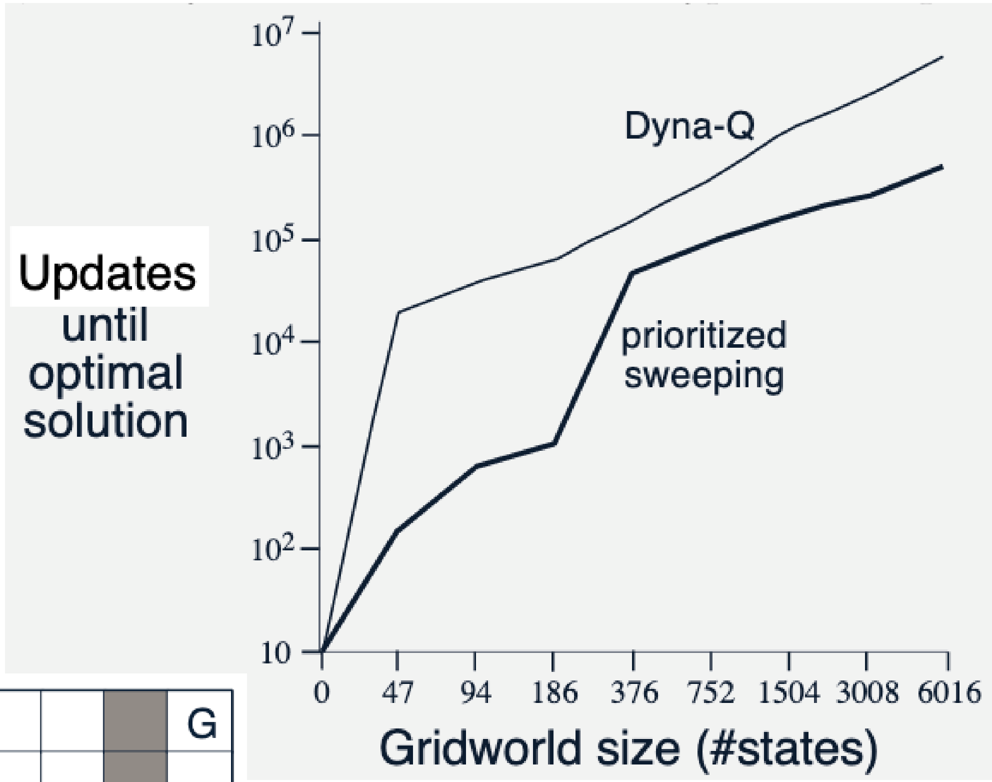
Prioritized sweeping for a deterministic environment

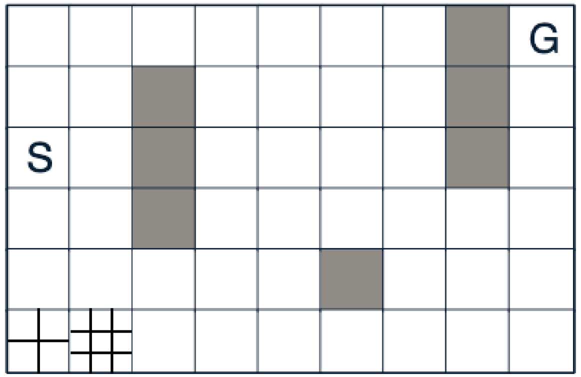
Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S, Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Model(S, A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S', a) - Q(S, A)|$.
(f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
(g) Repeat n times, while $PQueue$ is not empty:
    $S, A \leftarrow first(PQueue)$
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:
        $\bar{R} \leftarrow$ 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

Once S was updated check if and by how much all predecessor states need to be updated. Insert relevant transitions to reply buffer with the relevant priority

Prioritized sweeping

- 4-connected grid with varying resolution
- Deterministic action outcomes (N,E,S,W)
- +1 reward for moving into G
- +0 otherwise
- $\gamma = 0.95$
- Hyper parameters
  - $\alpha = 0.1$
  - $\varepsilon = 0.1$
  - $n = 5$





Stochastic environments

Prioritized sweeping for a ~~deterministic~~ environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S,Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Model(S,A) \leftarrow R, S'$
(e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
(f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
(g) Repeat n times, while $PQueue$ is not empty:
$S, A \leftarrow first(PQueue)$
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:
$\bar{R} \leftarrow$ 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

What adjustments are required for a stochastic environment?

Stochastic environments

Prioritized sweeping for a ~~deterministic~~ environment

Initialize $Q(s,a)$, $\boxed{Model(s,a)}$, for all s, a, and $PQueue$ to empty
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S,Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $\boxed{Model(S,A) \leftarrow R, S'}$
(e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
(f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
(g) Repeat n times, while $PQueue$ is not empty:
  $S, A \leftarrow first(PQueue)$
  $R, S' \leftarrow Model(S,A)$
  $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
  Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:
    $\bar{R} \leftarrow$ 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

Model will now return a distribution over S' and R

Stochastic environments

Prioritized sweeping for a ~~deterministic~~ environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow policy(S,Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Model(S,A) \leftarrow R, S'$
(e) $P \leftarrow |\boxed{R + \gamma \max_a Q(S',a)} - Q(S,A)|$.
(f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
(g) Repeat n times, while $PQueue$ is not empty:
$S, A \leftarrow first(PQueue)$
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha[\boxed{R + \gamma \max_a Q(S',a)} - Q(S,A)]$
Repeat, for all $\bar{S}, \bar{A}$ predicted to lead to S:
$\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
$P \leftarrow |\boxed{\bar{R} + \gamma \max_a Q(S,a)} - Q(\bar{S},\bar{A})|$.
if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

Replace update with expected value:

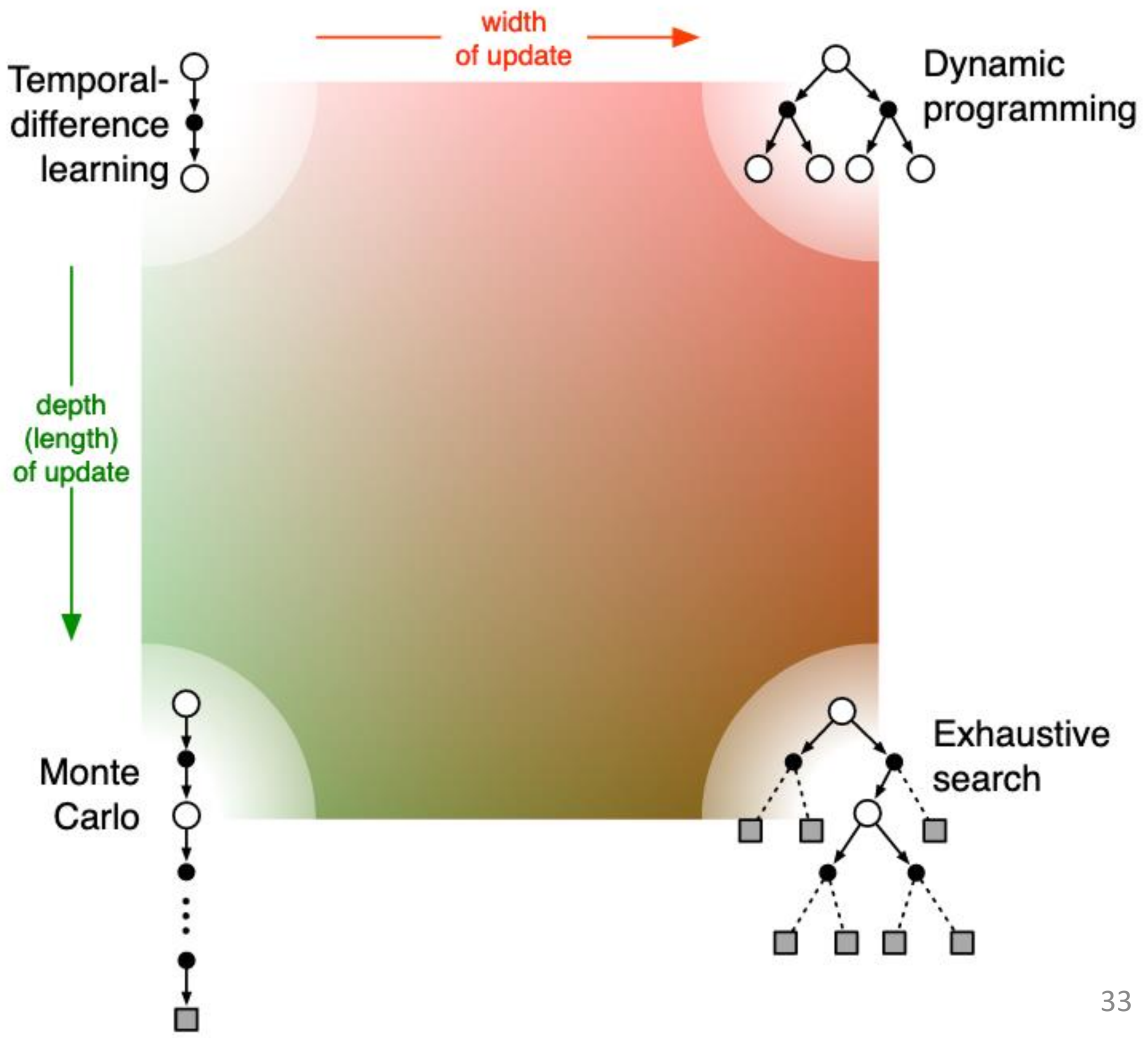
$$R + \gamma \sum_{s'} P(s'|S,A) \max_a Q(s',a)$$

What did we learn?

- Learning an approx. model alongside any online algorithm is free
- The approx. model can be used to speed up state or action value learning -- especially for sparse reward settings and off-policy learning
- On the other hand, errors in the approx. model can lead to inaccurate updates
- Longer rollouts lead to faster learning but result in large prediction errors
- When setting the *n* hyper parameter in Dyna-Q, take into account model accuracy and cost of real vs simulated transition

Summary of tabular learning approaches

- Dynamic programing
  - Model based
  - Offline
- Exhaustive search (Expectimax)
  - Model based
  - Offline
  - NP-C
- Temporal difference
  - Model free
  - Online
  - Off policy
  - Slow value propagation (low variance)
- Monte Carlo
  - Model free
  - Online
  - On policy
  - Fast value propagation (high variance)



Important concepts

- Definition of return (discounted sum of rewards)
  - Is the task episodic or continuing, discounted or not
- Action (q) vs. State (v) value
  - What kind of values should be estimated? If only state values are estimated, then either a model or a separate policy (as in actor–critic methods) is required for action selection
- Action selection/exploration
  - How are actions selected to ensure a suitable trade-off between exploration and exploitation? We have considered several approaches: ε-greedy, optimistic initialization of values, softmax, and upper confidence bound

Important concepts

- Synchronous vs. asynchronous
  - Are the updates for all states performed simultaneously or one by one in some order?
- Real vs. simulated
  - Should one update be based on real experience or simulated experience? If both, how much of each?
- Location of updates
  - What states or state–action pairs should be updated? Model-free methods can choose only among the states and state–action pairs actually encountered

What next?

- **Lecture**: On-policy Prediction with Approximation
- **Project**:
  - Finalize your project proposal, due Sep 30