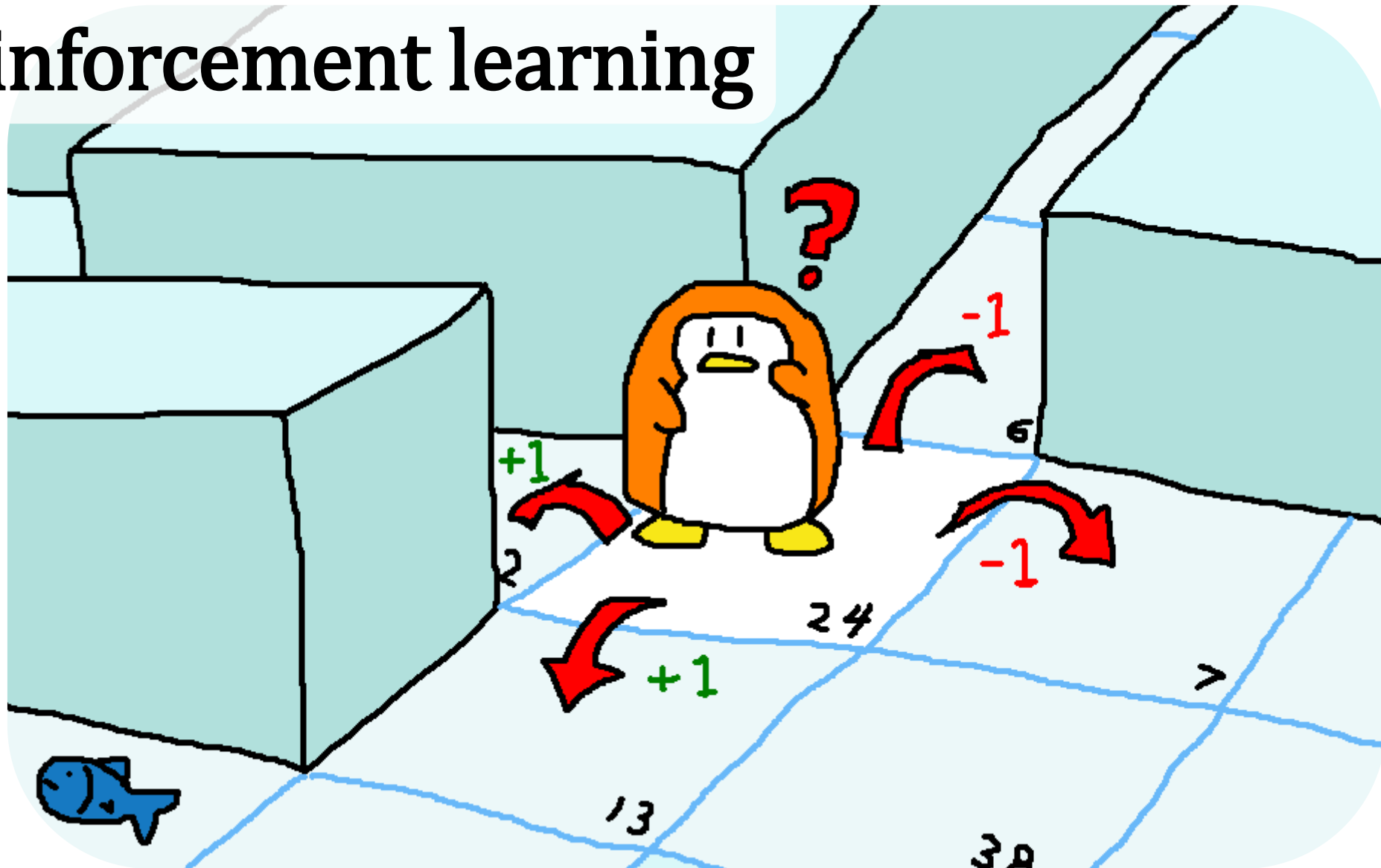


Finite-Horizon Temporal Difference Methods

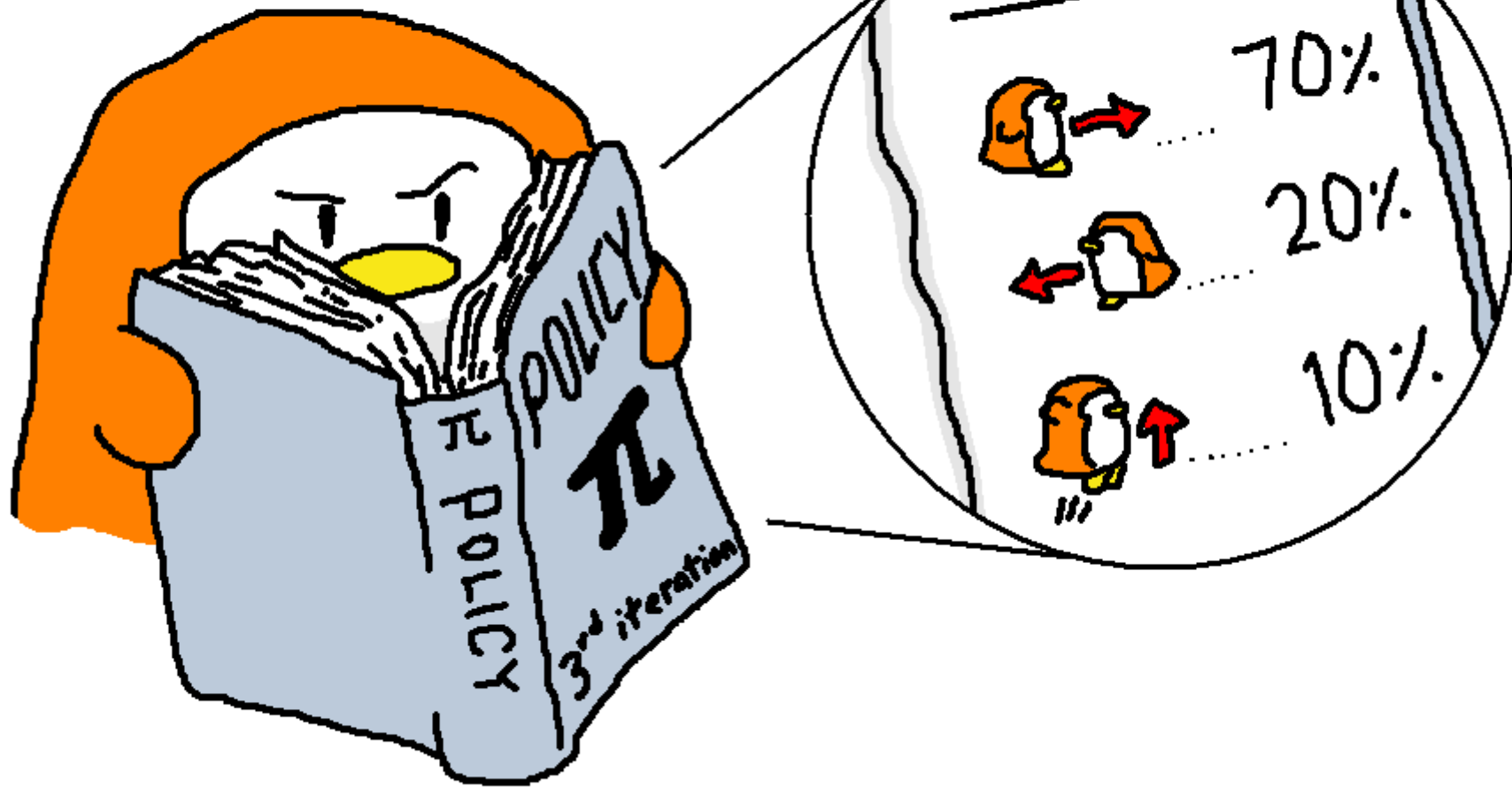


KRIS DE ASIS

Reinforcement learning



Policies



Returns and value functions

Return:

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

Value functions:

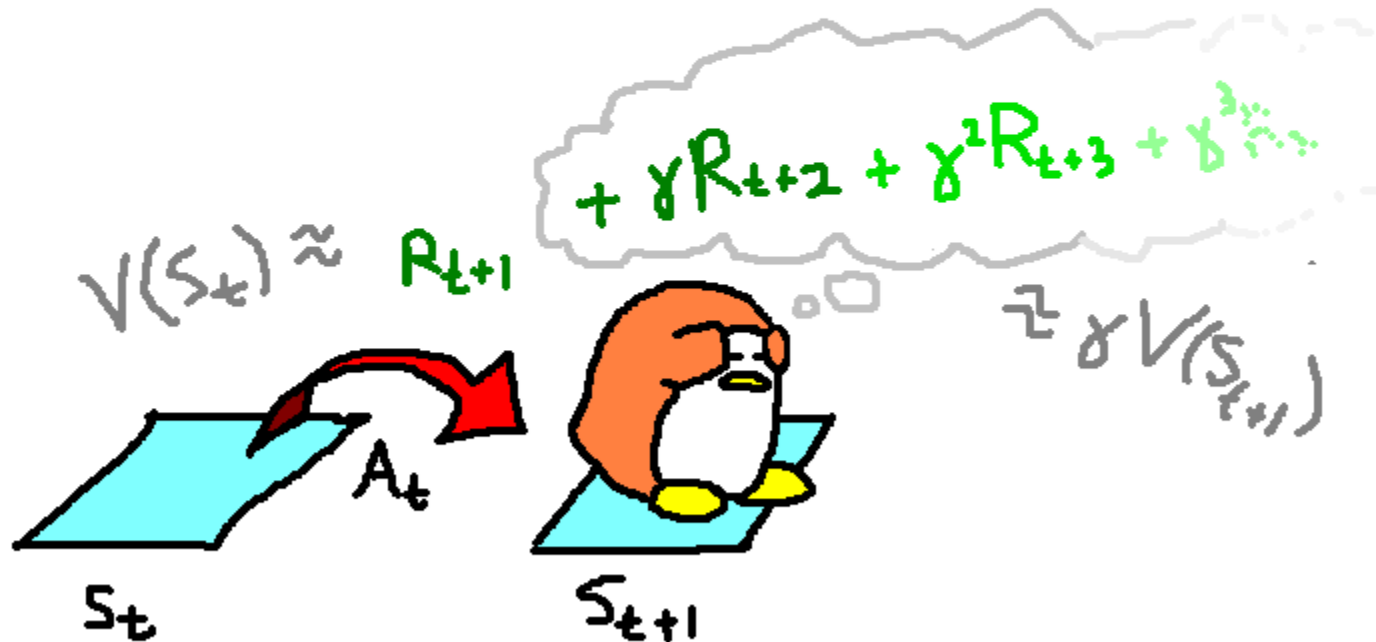
$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s]$$

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

TD learning

Approximate value function:

$$V \approx v_{\pi}$$



TD learning

Approximate value function:

$$V \approx v_\pi$$

TD target:

$$\hat{G}_t = R_{t+1} + \gamma V(S_{t+1})$$

TD update:

$$V(S_t) \leftarrow V(S_t) + \alpha [\hat{G}_t - V(S_t)]$$

Can be applied to any signal of interest to learn GVFs

Value function update target depends on itself- can learn **independent of span**

Finite-horizon returns

$$R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + R_{t+5} \left. \vphantom{R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + R_{t+5}} \right\} + R_{t+6} + R_{t+7} + R_{t+8} + \dots$$

A case of **time-dependent** discounting

Episodic problems can produce finite-horizon returns, but:

- The horizon depends on the environment
- Can still be infinite if a policy avoids terminal states

Finite-horizon returns

Finite-horizon return:

$$G_t^h \stackrel{\text{def}}{=} R_{t+1} + R_{t+2} + \cdots + R_{t+h} = \sum_{k=1}^{\min(h, T-t)-1} R_{t+k+1}$$

Value functions:

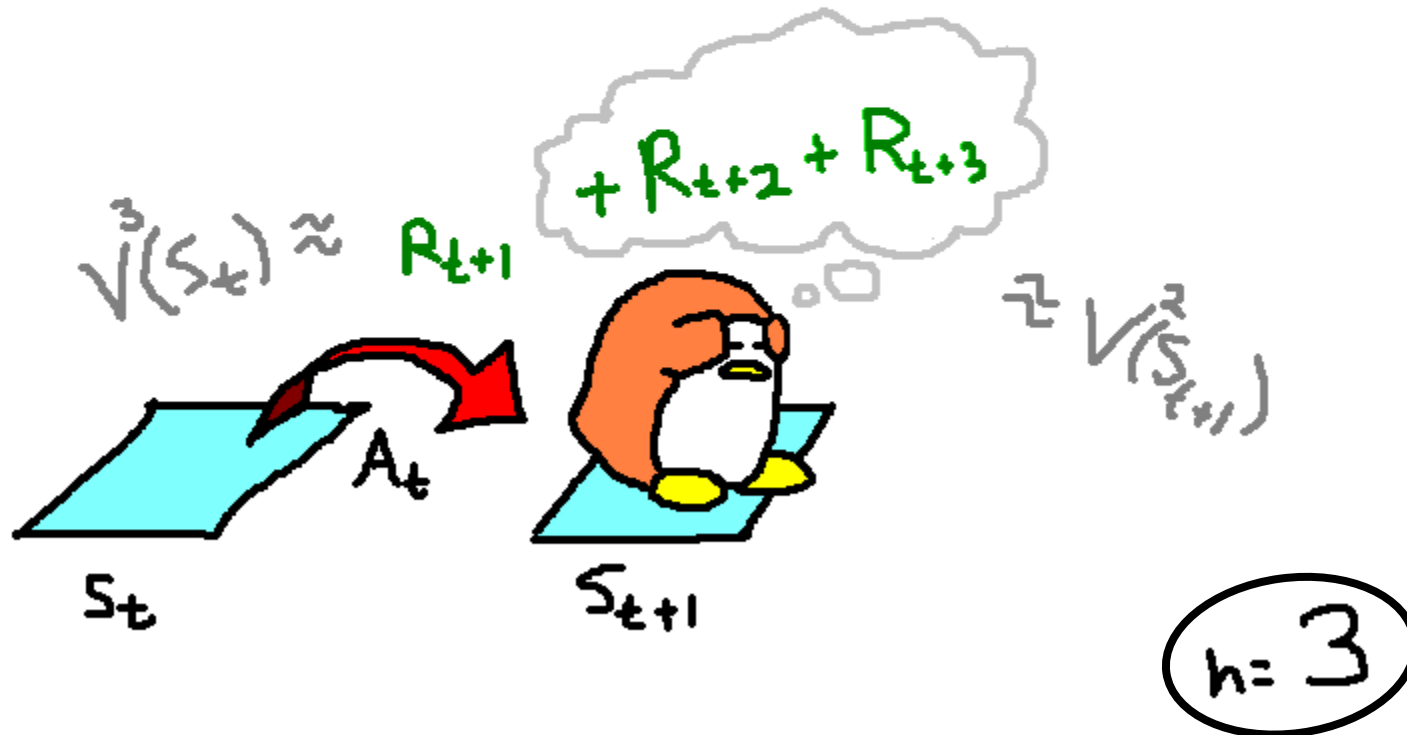
$$v_{\pi}^h(s) \stackrel{\text{def}}{=} \mathbb{E}_{\pi} [G_t^h | S_t = s]$$

$$q_{\pi}^h(s, a) \stackrel{\text{def}}{=} \mathbb{E}_{\pi} [G_t^h | S_t = s, A_t = a]$$

One-step finite-horizon TD (FHTD)

Approximate value function:

$$V^h \approx v_\pi^h$$



One-step finite-horizon TD (FHTD)

Approximate value functions- for each $h \in \{1, 2, 3, \dots H\}$:

$$V^h \approx v_\pi^h$$

TD targets- for each $h \in \{1, 2, 3, \dots H\}$:

$$\hat{G}_t^h = R_{t+1} + V^{h-1}(S_{t+1})$$

TD updates- for each $h \in \{1, 2, 3, \dots H\}$:

$$V^h(S_t) \leftarrow V^h(S_t) + \alpha [\hat{G}_t^h - V^h(S_t)]$$

$$V^0(s) = 0 \quad \forall s$$

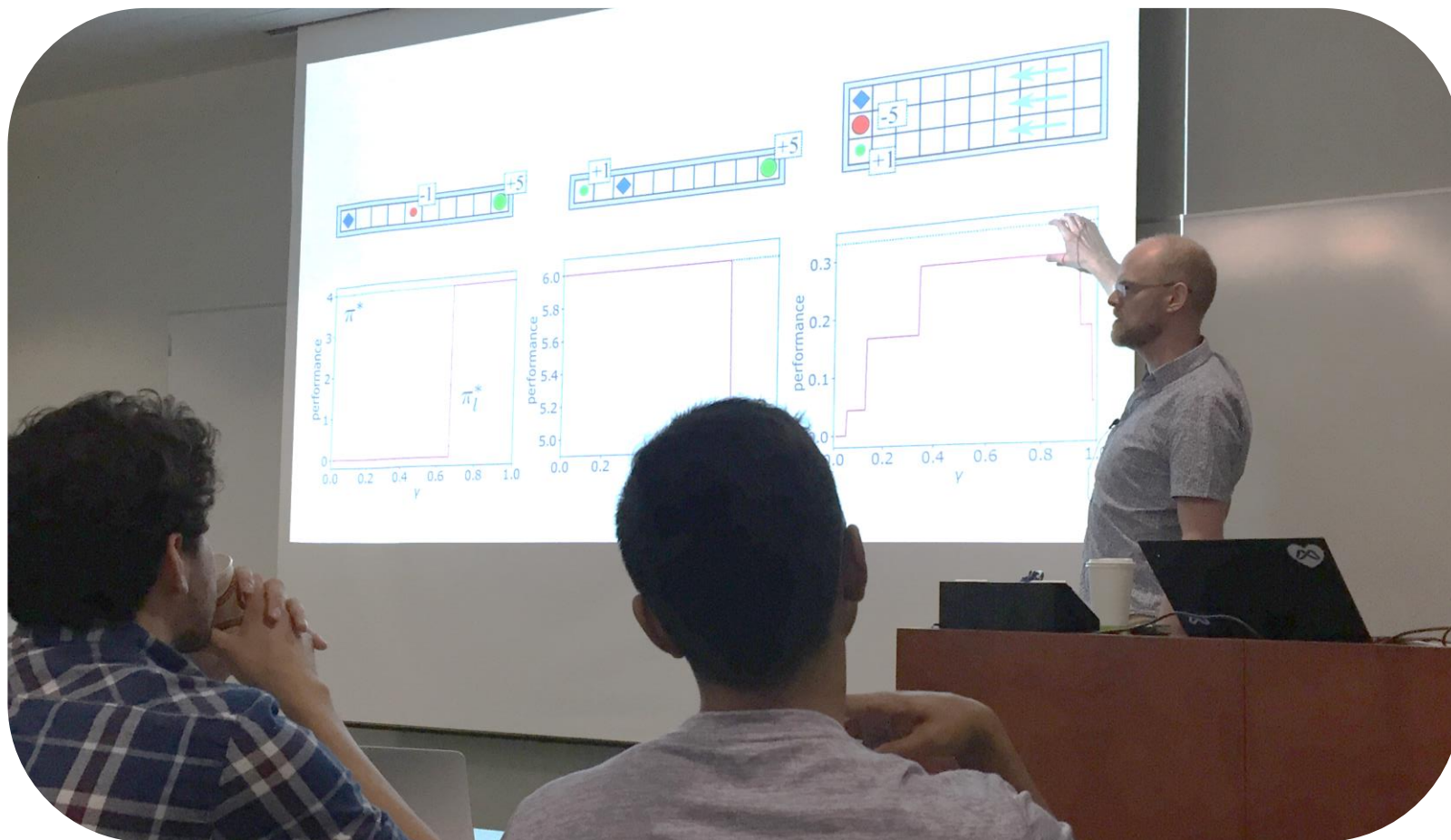
Sutton (1988)

5.3 Prediction by a fixed interval

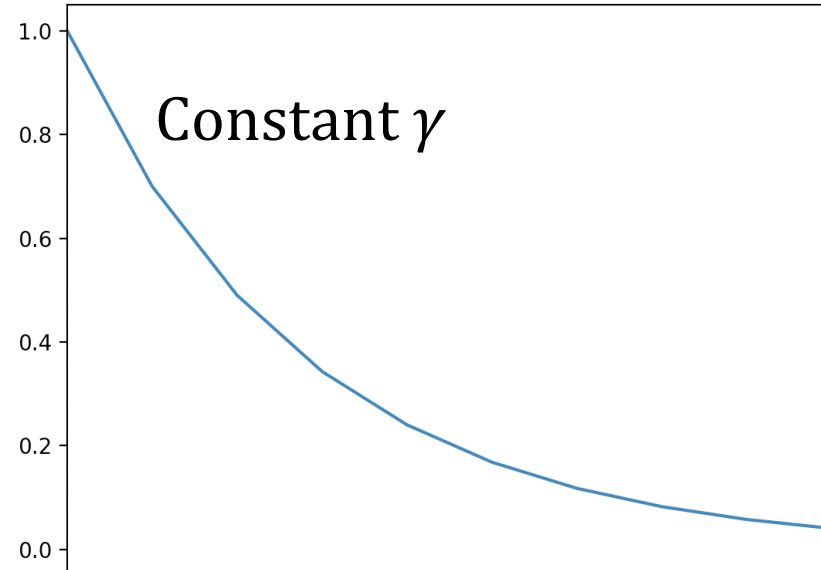
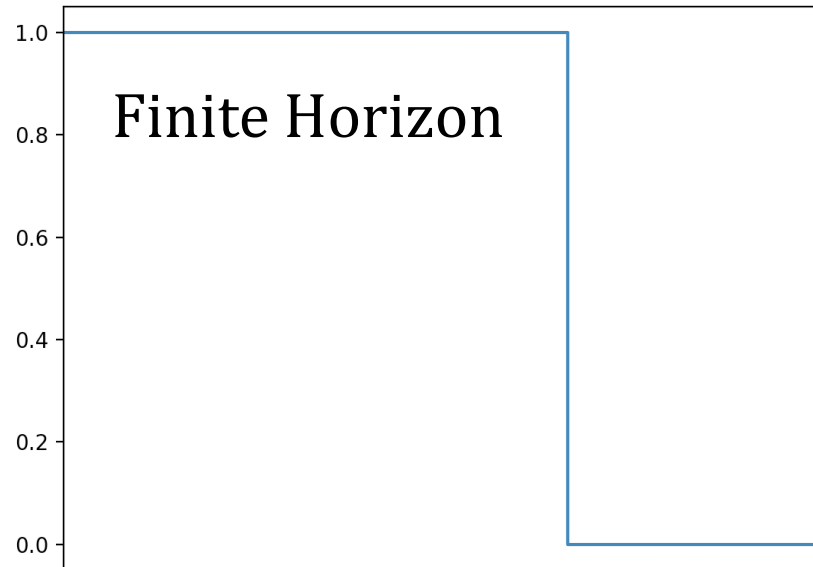
Finally, consider the problem of making a prediction for a particular fixed amount of time later. For example, suppose you are interested in predicting one week in advance whether or not it will rain - on each Monday, you predict whether it will rain on the following Monday, on each Tuesday, you predict whether it will rain on the following Tuesday, and so on for each day of the week. Although this problem involves a sequence of predictions, TD methods cannot be directly applied because each prediction is of a different event and thus there is no clear desired relationship between them.

In order to apply TD methods, this problem must be embedded within a larger family of prediction problems. At each day t , we must form not only P_t^7 , our estimate of the probability of rain seven days later, but also P_t^6 , P_t^5 , ..., P_t^1 , where each P_t^δ is an estimate of the probability of rain δ days later. This will provide for overlapping sequences of inter-related predictions, e.g.,

van Seijen (-2 days)

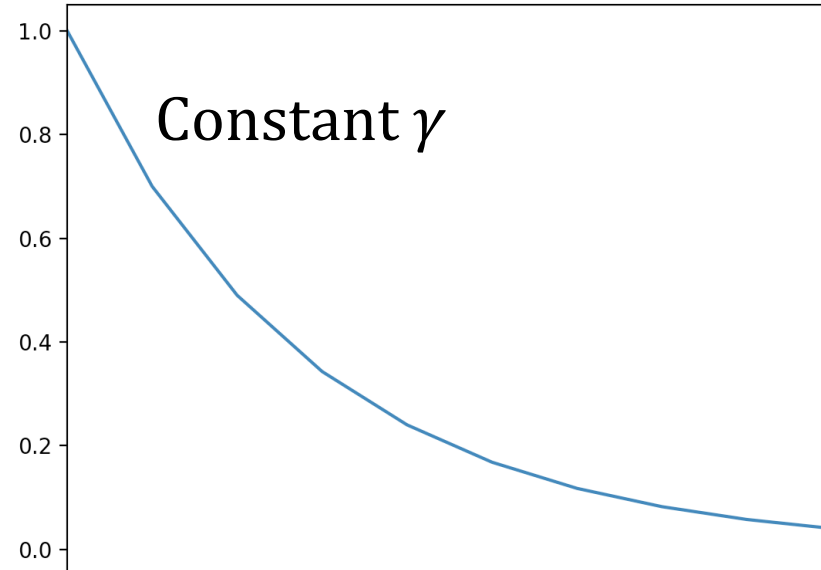
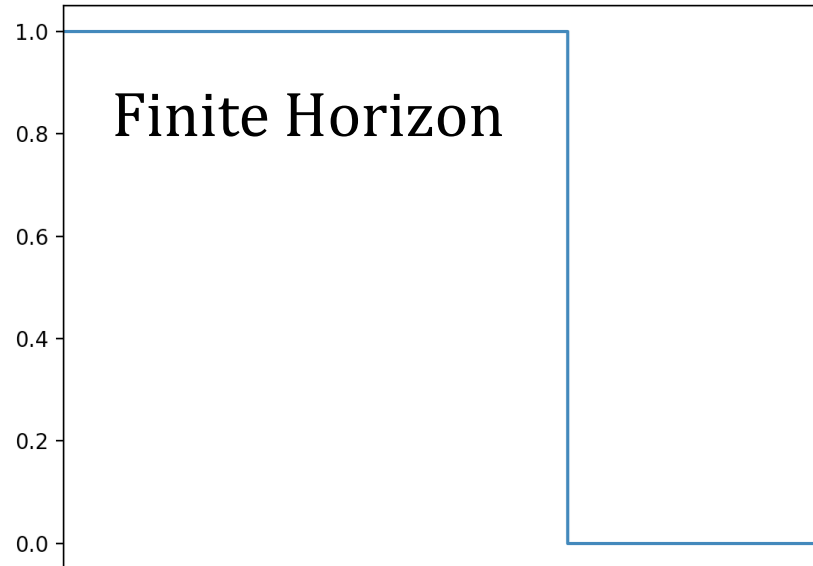


Step-functions and filters of the return



Can subtract step functions to get an impulse at a particular time step- can extract the expected reward for any future time step.

Step-functions and filters of the return



If interpreted as a filter, geometric weights have a non-linear phase response, distorting the underlying signal

Compositional GVFs

A simple case of compositional GVFs

A clear, interpretable example of specifying a question that can leverage what was already learned:

- If we already learned what happens in 8 steps, we could specify a question about what happens in 9 steps, and not start from scratch

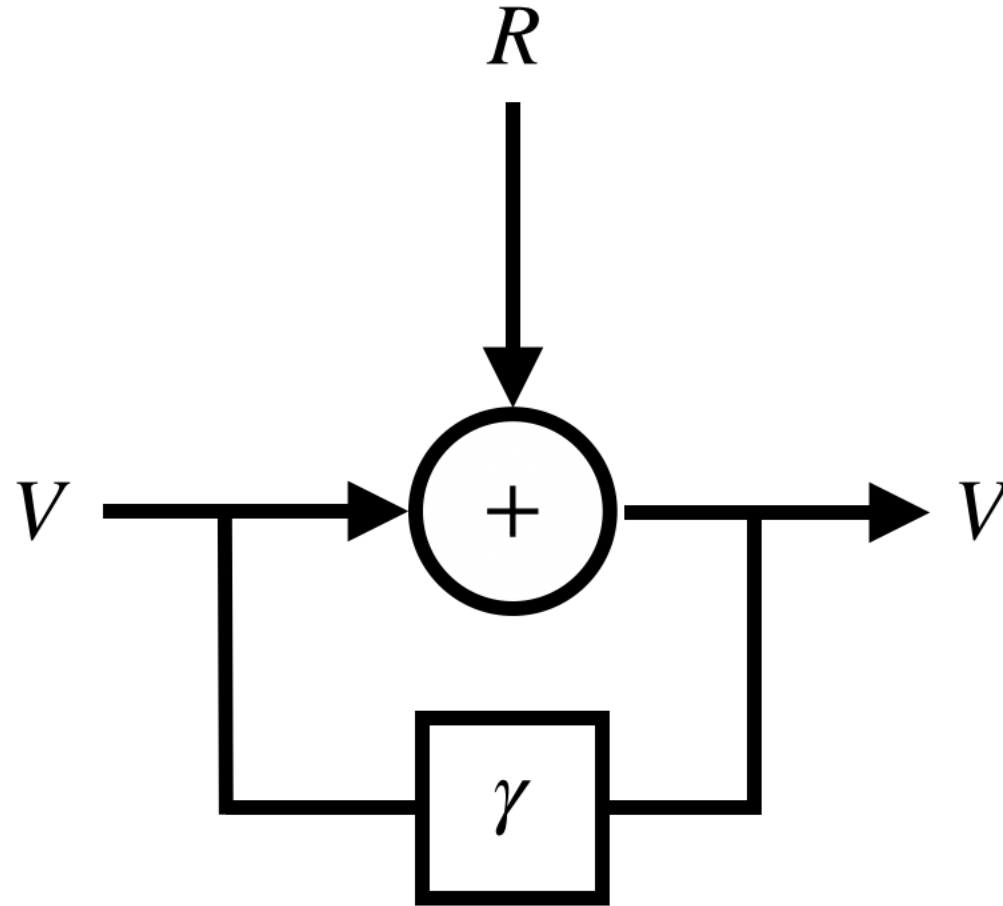
Suboptimal control

Trying to maximize over a fixed, shorter horizon won't be optimal

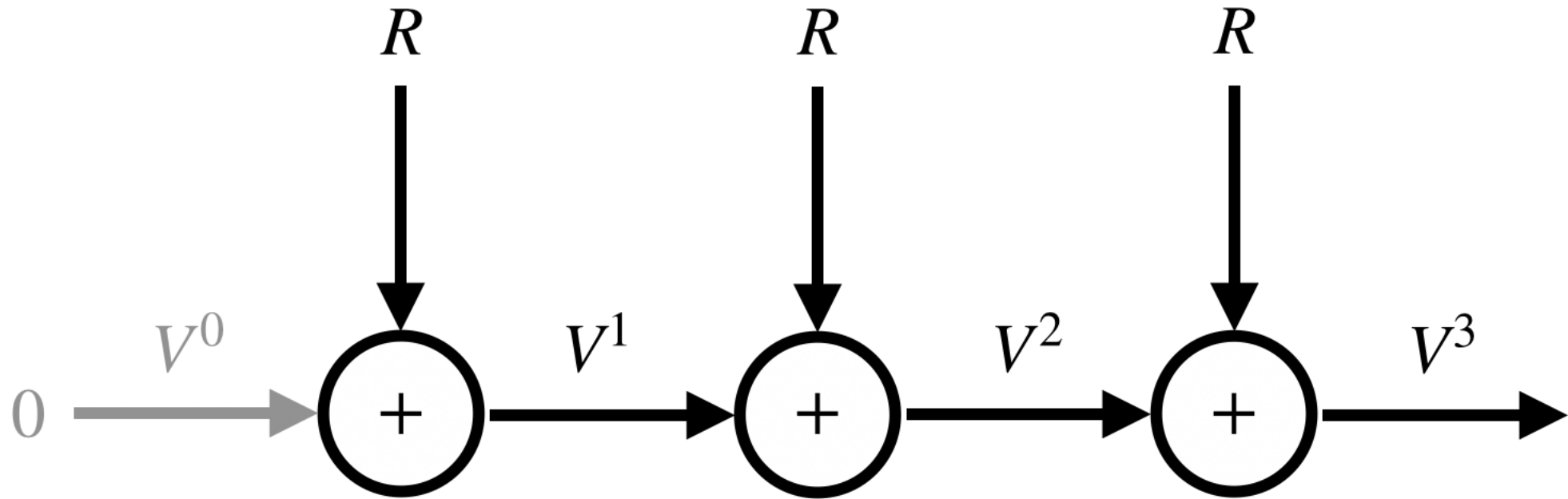
However, optimality isn't guaranteed anyways when values are only approximated (i.e. function approximation, constant step sizes)

Can continually increase the final horizon, leveraging what has already been learned, until sufficiently optimal

Unrolling TD



Unrolling TD



Decoupled bootstrapping

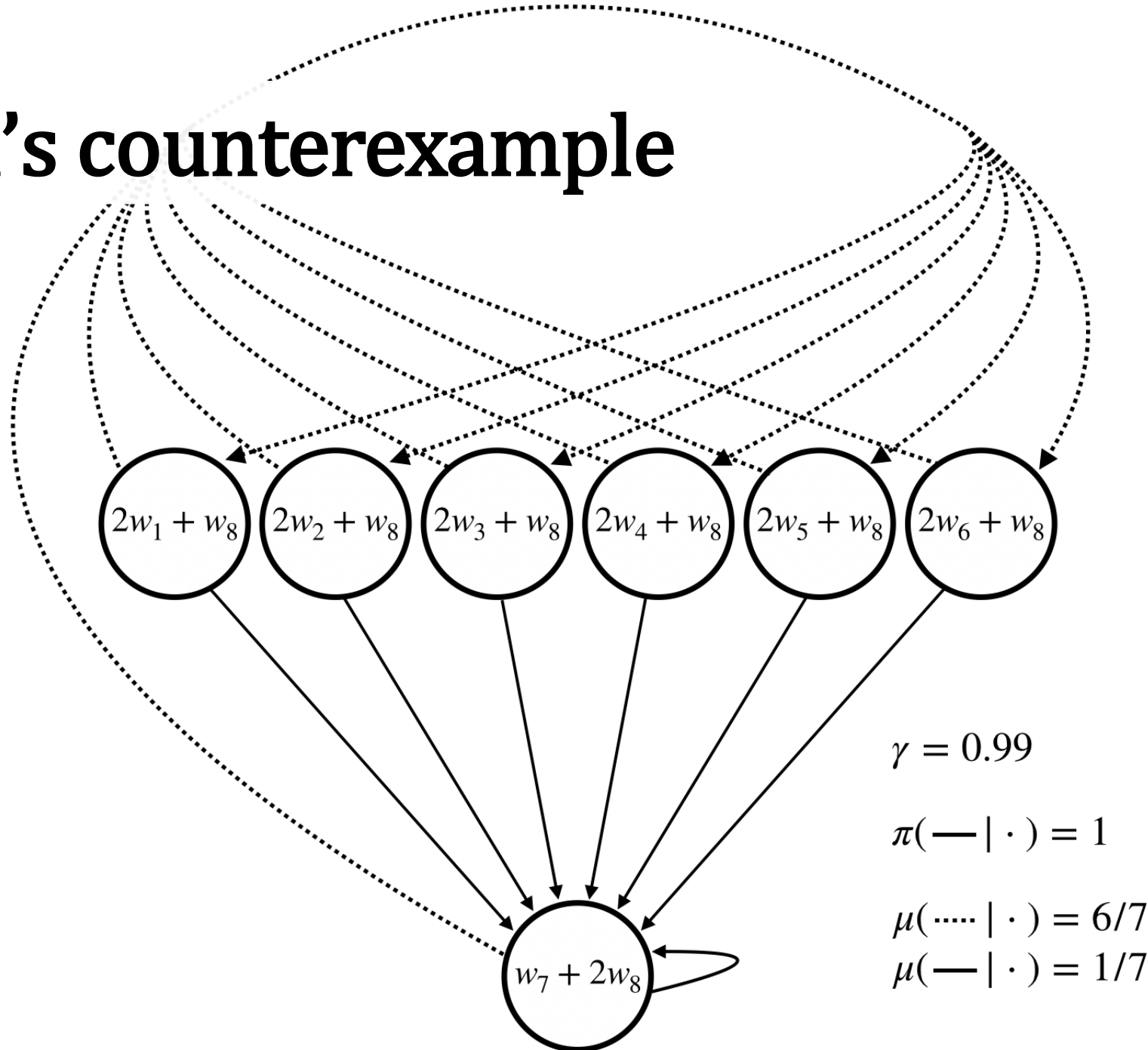
V^1 learns the immediate reward, a stable (Monte Carlo) target

After V^1 converges, V^2 will have a stable target and eventually converge, then V^3 will have a stable target...

Appears compatible with **non-linear function approximation** in prediction under fixed policies

A way to bootstrap that **avoids the deadly triad?**

Baird's counterexample



Baird's counterexample

Predicted up to $H = \frac{1}{1-\gamma} = 100$

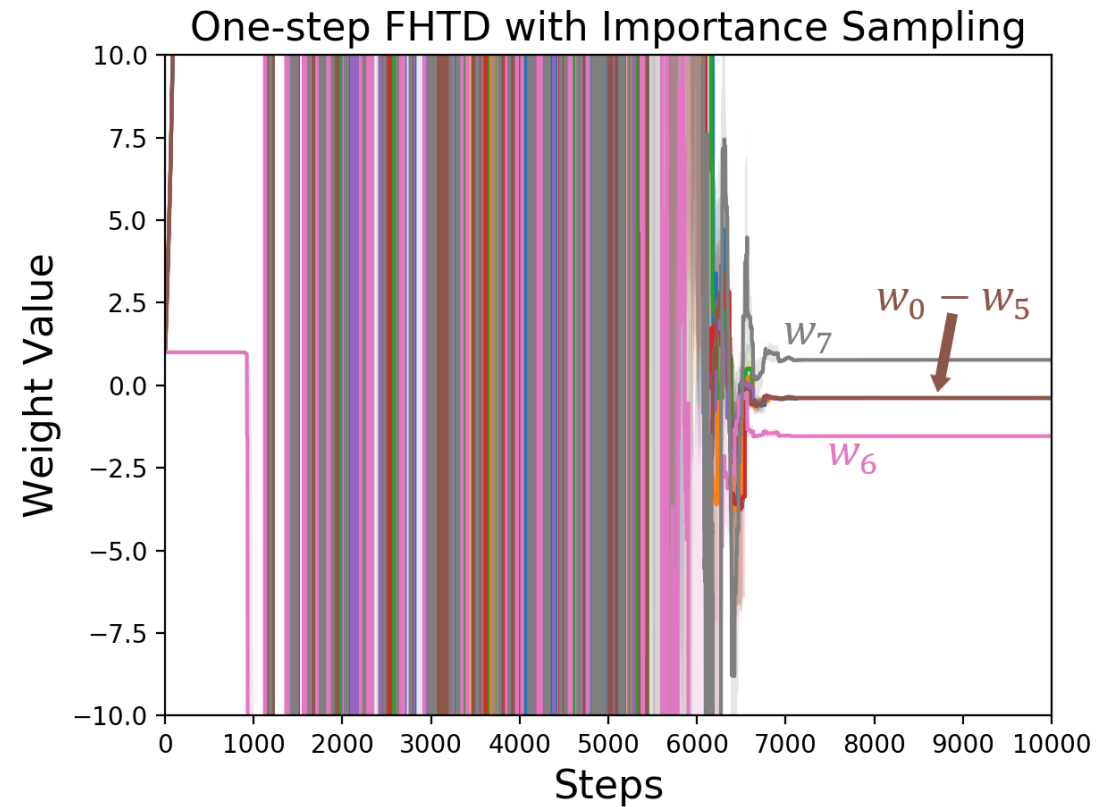
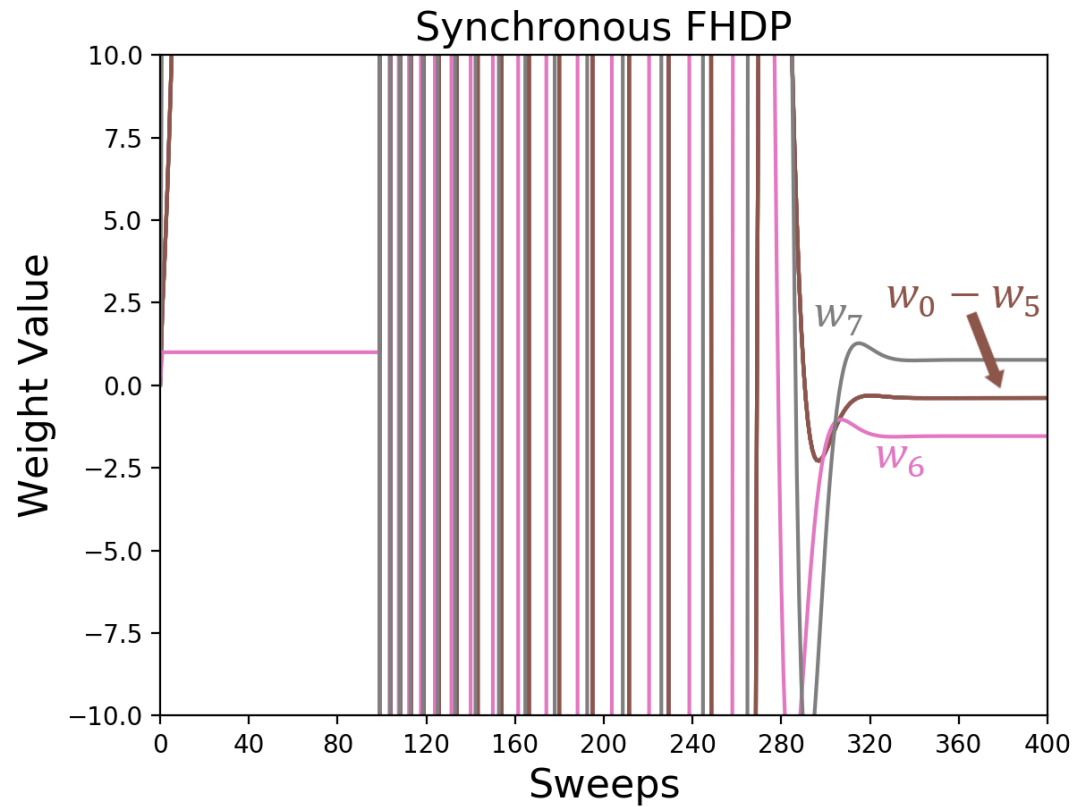
$$\mathbf{w}^h = [1, 1, 1, 1, 1, 1, 10, 1]^T \quad \forall h \in \{1, 2, 3, \dots, 100\}$$

Used finite-horizon dynamic programming:

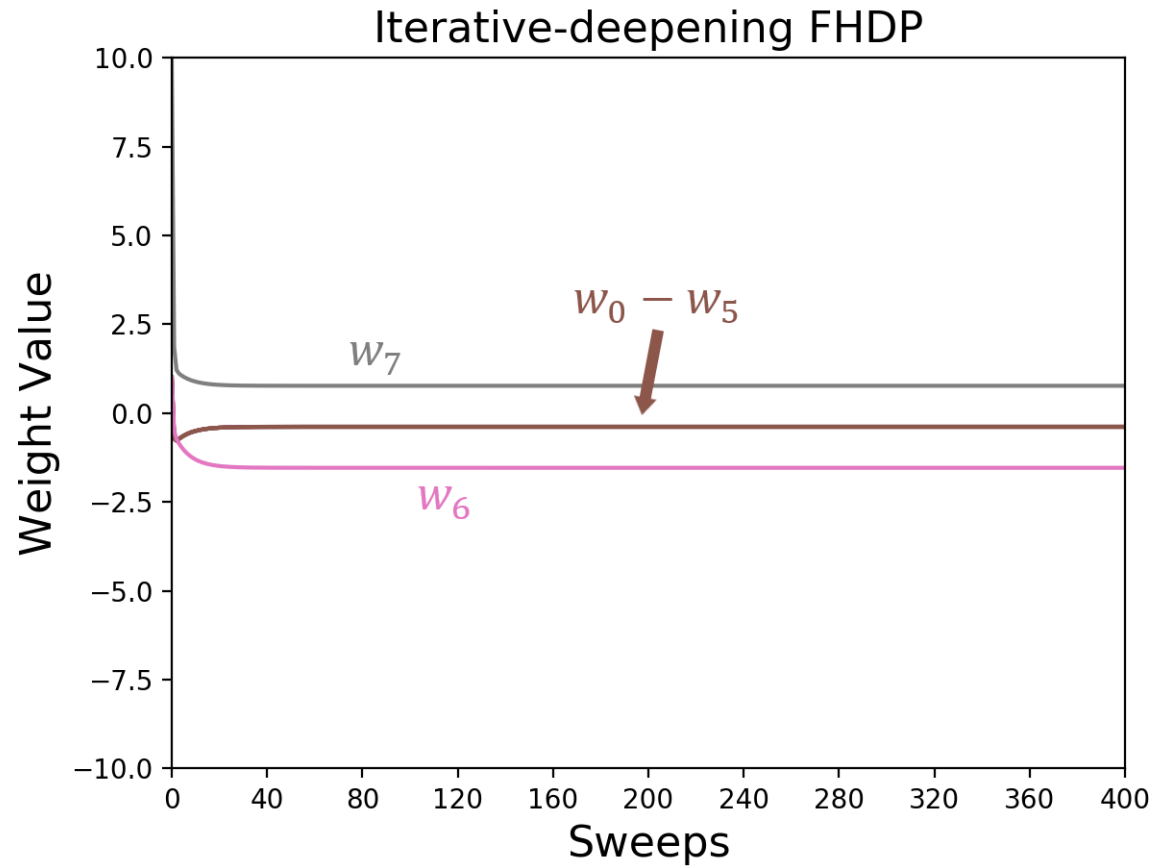
- Synchronous updating
- Iterative deepening

Also ran one-step FHTD with importance sampling

Baird's counterexample



Baird's counterexample



Not independent of span

Have to learn H value functions, computation scales **linearly** with the final horizon H

What about finite-horizon Monte Carlo?

- Stores last H rewards, but if you only care about the final horizon...
- Only has to update **one** value function!

n-step TD to the Rescue

1-step TD target:

$$\hat{G}_t = R_{t+1} + \gamma V(S_{t+1})$$

2-step TD target:

$$\hat{G}_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

3-step TD target:

$$\hat{G}_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})$$

n-step TD to the Rescue

1-step FHTD target:

$$\hat{G}_t^h = R_{t+1} + V^{h-1}(S_{t+1})$$

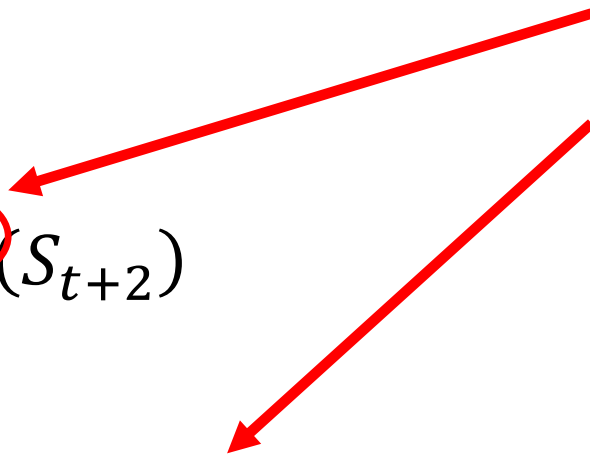
2-step FHTD target:

$$\hat{G}_t^h = R_{t+1} + R_{t+2} + V^{h-2}(S_{t+2})$$

3-step FHTD target:

$$\hat{G}_t^h = R_{t+1} + R_{t+2} + R_{t+3} + V^{h-3}(S_{t+3})$$

Skips value functions-
only have to learn $\left\lfloor \frac{H}{n} \right\rfloor$
of them!



Sub-linear in Span

Has to add n rewards, and update $\frac{H}{n}$ value functions. Computation scales with:

$$n - 1 + \frac{H}{n}$$

Worst case: H operations at $n = 1$ (one-step) and $n = H$ (MC)

Best case: $2\sqrt{H} - 1$ operations at $n = \sqrt{H}$

Sub-linear in Span

Example: 10-step FHTD to predict V^{100} :

- Store and sum last 10 rewards (Denoted ΣR)
- Estimate 10 value functions ($V^{10}, V^{20}, V^{30}, \dots, V^{100}$)

$$V^{10} \approx \Sigma R + V^0$$

$$V^{20} \approx \Sigma R + V^{10}$$

⋮

$$V^{100} \approx \Sigma R + V^{90}$$

FHTD Control?

Trivial to extend multi-step FHTD for learning action-values in **prediction**, less trivial for **control**

Each horizon has to be greedy with respect to themselves

The greedy action of one horizon can differ from the greedy action of another horizon- FHTD control is inherently **off-policy**

FHTD Control?

The savings from n-step may not be possible without approximations- need to know what the greedy action is for each horizon, so we can't skip over learning them ☹️

The optimal policy of a horizon is unaffected by policy improvement of later horizons- can leverage previously learned policies! 😊

One-step finite-horizon Q-learning

Approximate value functions- for each $h \in \{1, 2, 3, \dots H\}$:

$$Q^h \approx q_*^h$$

TD targets- for each $h \in \{1, 2, 3, \dots H\}$:

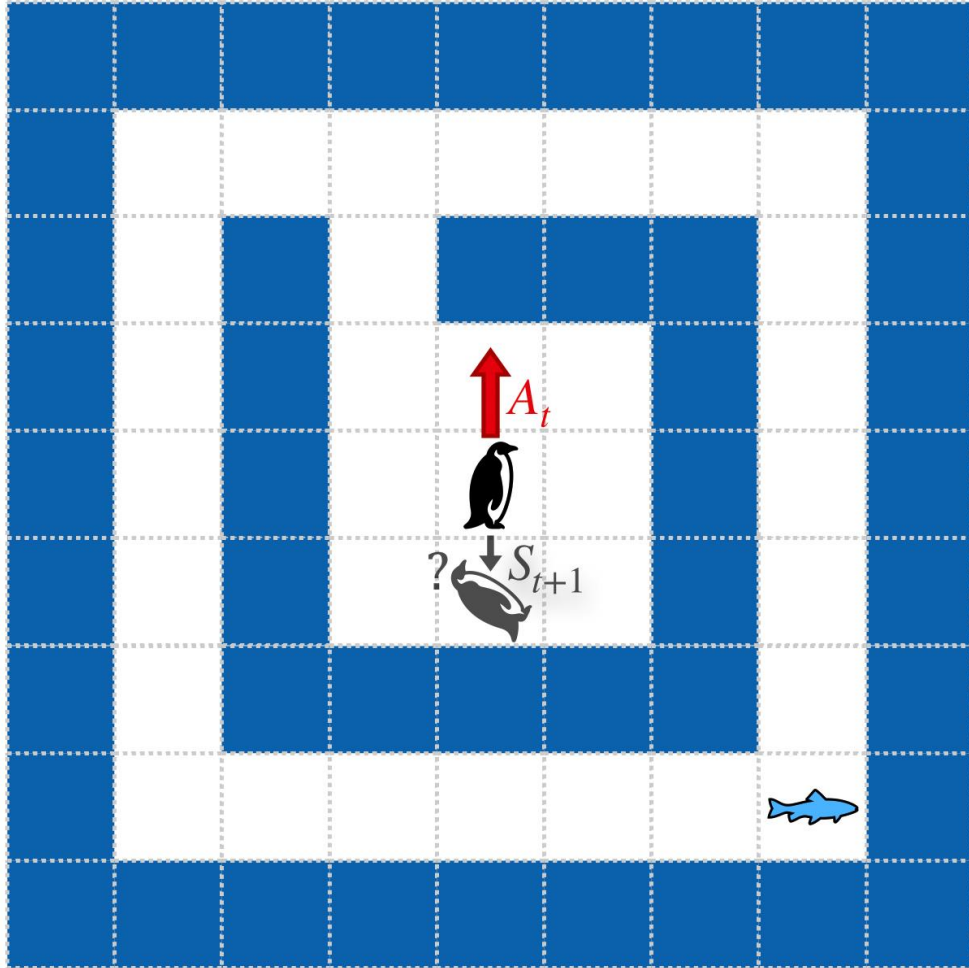
$$\hat{G}_t^h = R_{t+1} + \max_{a'} Q^{h-1}(S_{t+1}, a')$$

TD updates- for each $h \in \{1, 2, 3, \dots H\}$:

$$Q^h(S_t, A_t) \leftarrow Q^h(S_t, A_t) + \alpha [\hat{G}_t^h - Q^h(S_t, A_t)]$$

$$Q^0(s, a) = 0 \quad \forall s, a$$

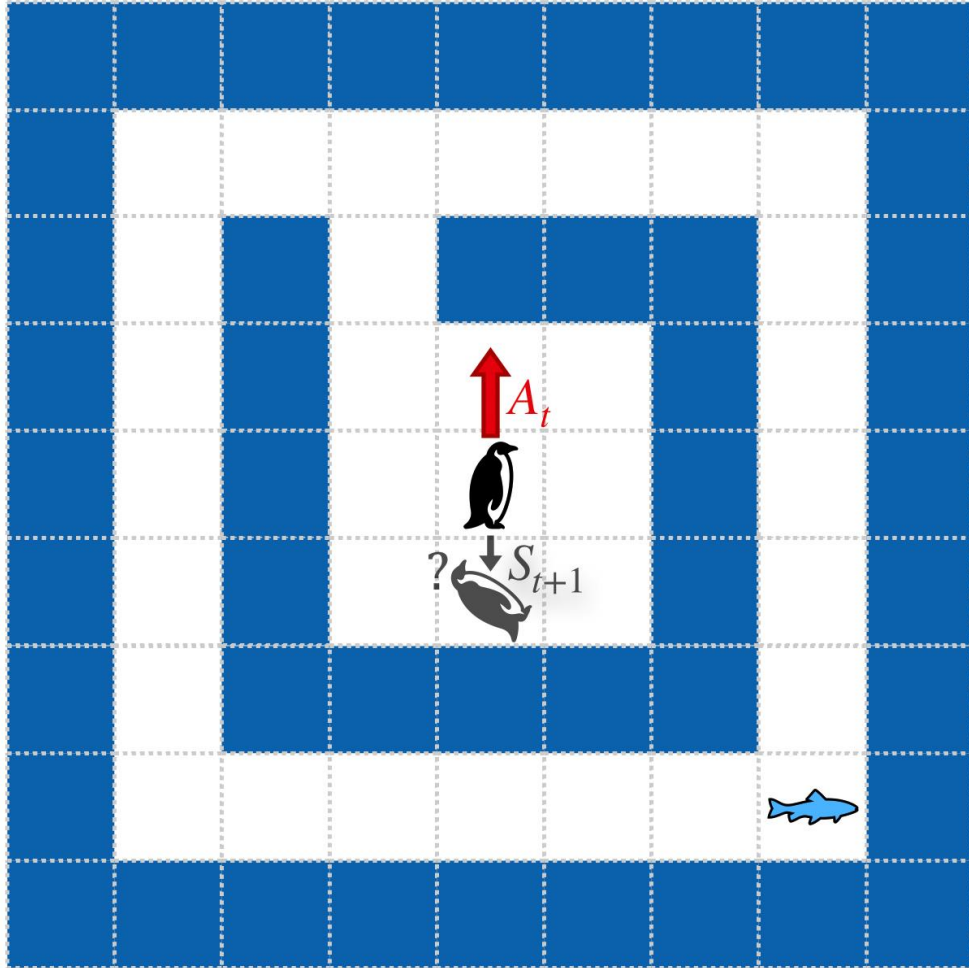
FHTD control



Slippery maze:

- Episodic grid world
- 4-directional movement, 75% chance action gets overridden with random action
- Reward of -1 at each step

FHTD control



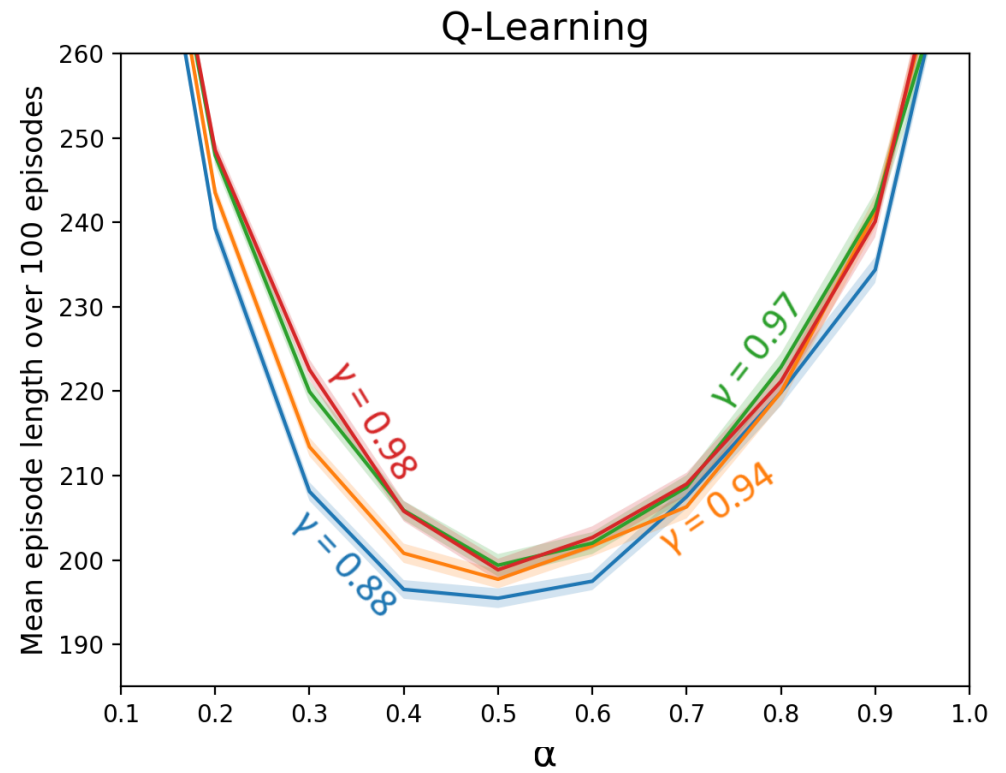
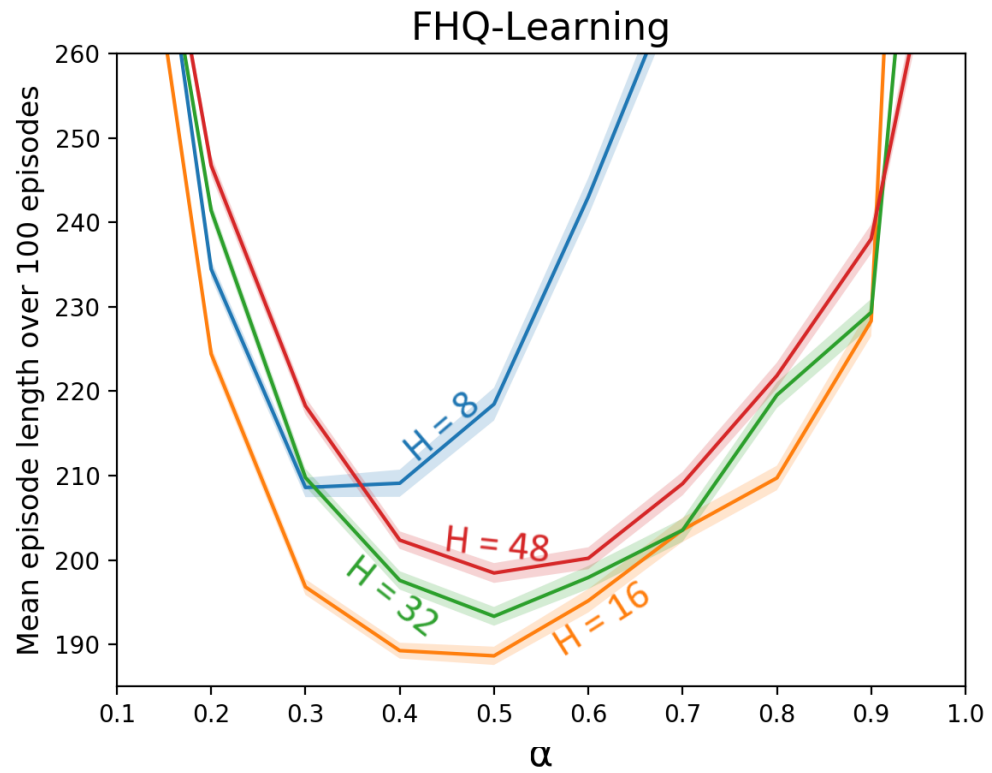
FHQ-learning:

- Behaved ϵ -greedy w.r.t. Q^H ($\epsilon = 0.1$)
- Compared $H \in \{8, 16, 32, 48\}$

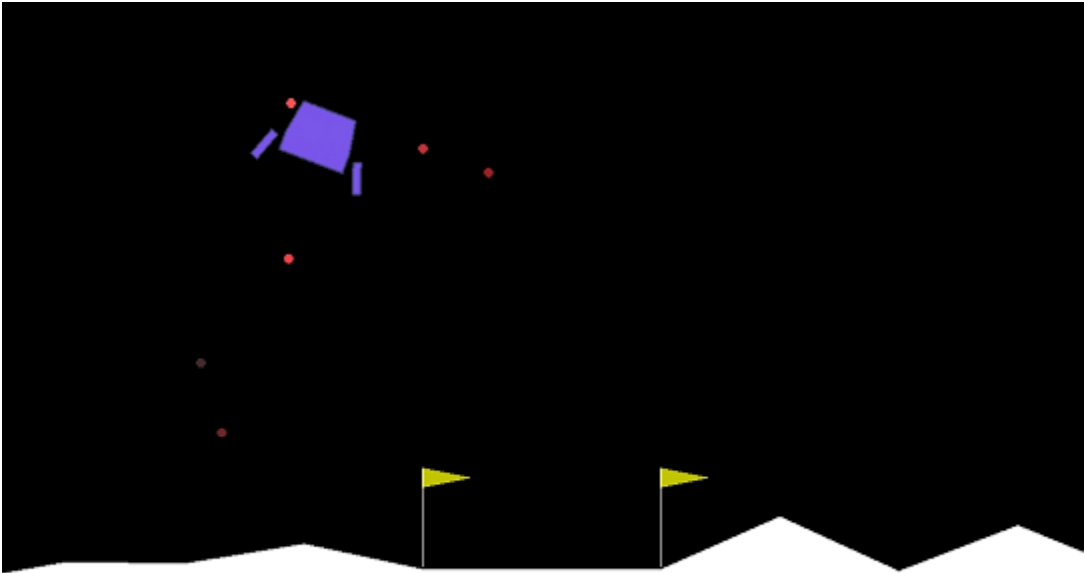
Q-learning:

- ϵ -greedy w.r.t. Q ($\epsilon = 0.1$)
- Compared $\gamma = 1 - \frac{1}{H}$

FHTD control

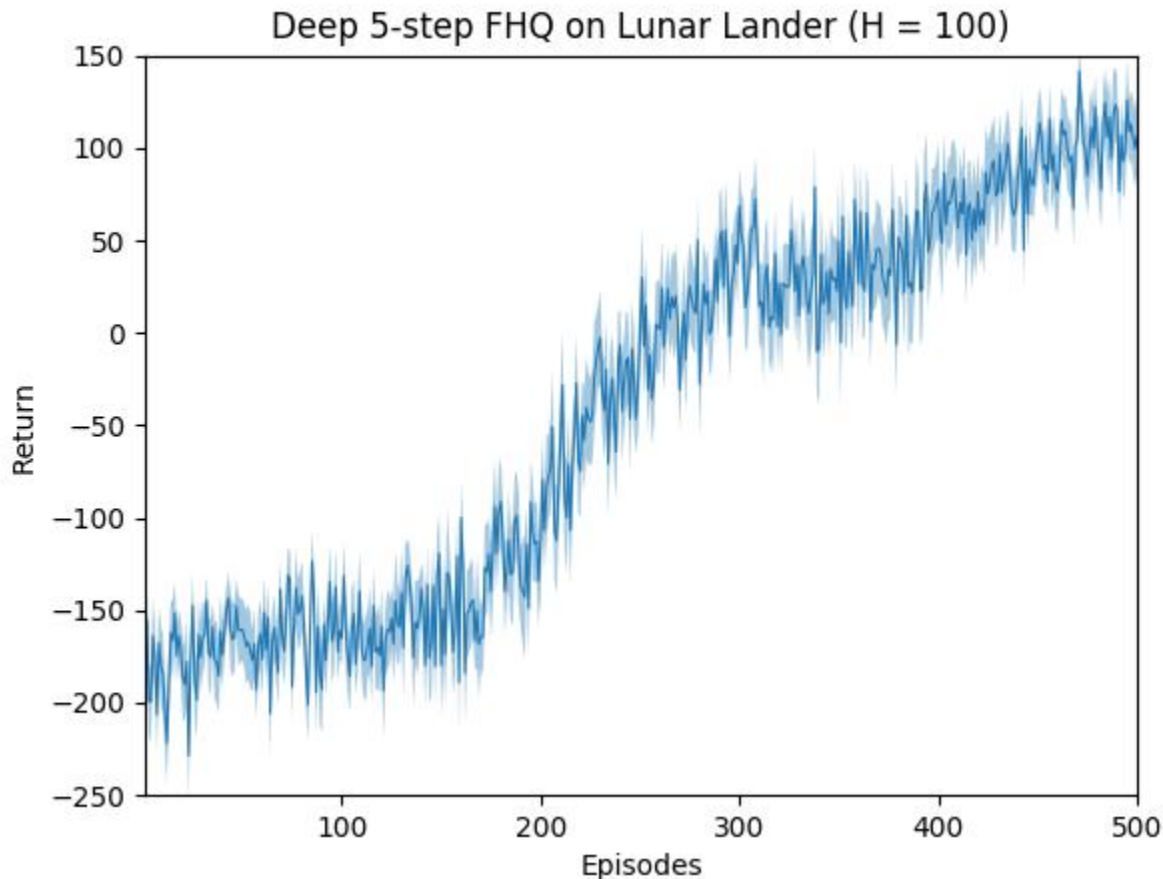


Deep n-step FHQ ($n = 5, H=100$)



- Lunar Lander environment
- Fully connected, one hidden layer
- Horizons treated as heads over a shared representation

Deep n-step FHQ ($n = 5, H=100$)



- **No** momentum
- **No** target network
- **No** experience replay
- Will probably still improve with the above

Summary

Might be useful to only considering a shorter, finite-horizon:

- Stable update targets
- Can get exact notion of what happens and when
- Clear example of compositional GVFs
- Empirical benefits

No longer independent of span, but can reduce computation in prediction with n-step methods

New horizons to pursue

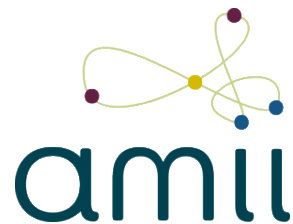


Better ways to handle unavoidable off-policyness of FHTD control?
Approximations for reducing computation in control?

Subtracting subsequent horizons allows for extracting expected individual rewards- can maybe learn and plan with an automatically unrolled model in time?

Online iterative-deepening? Maybe each horizon can have its own step size, and use something like TIDBD (Kearney et al., 2018)?

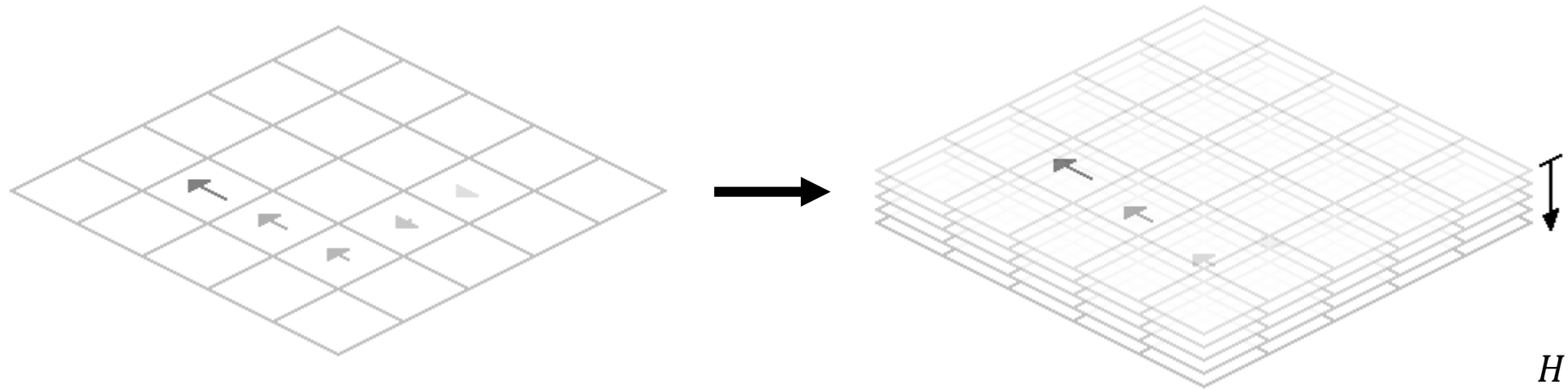
Questions?



KRIS DE ASIS

What about TD(λ)?

Results in truncated traces that travel through horizons

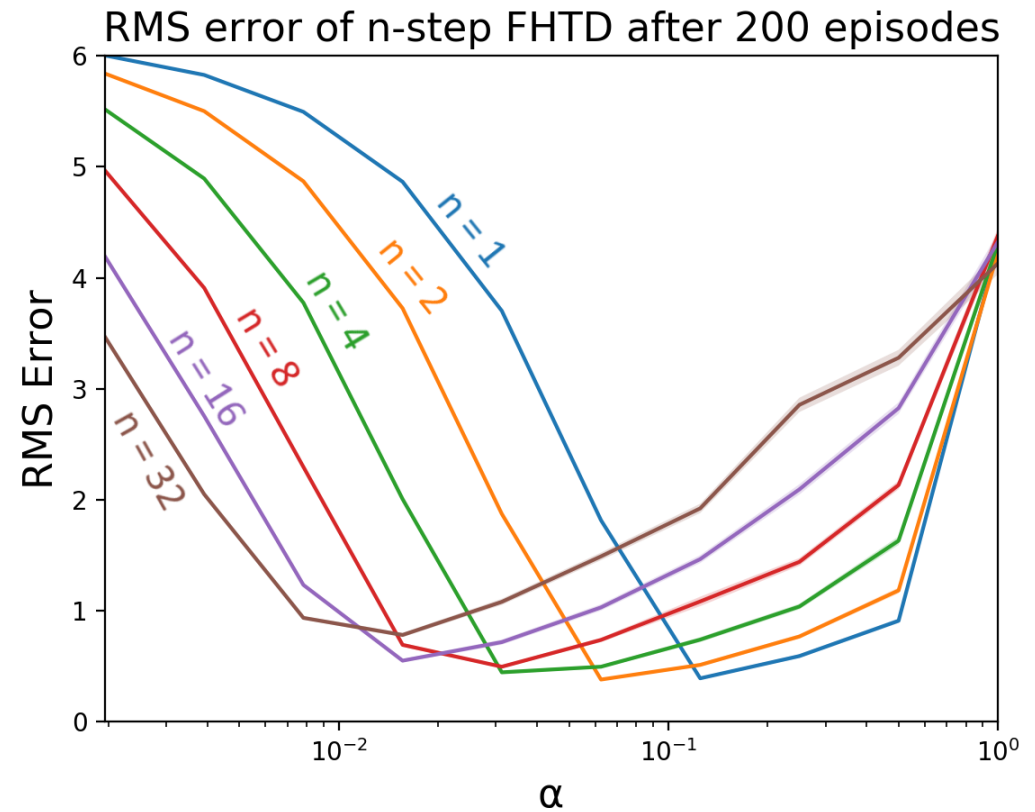
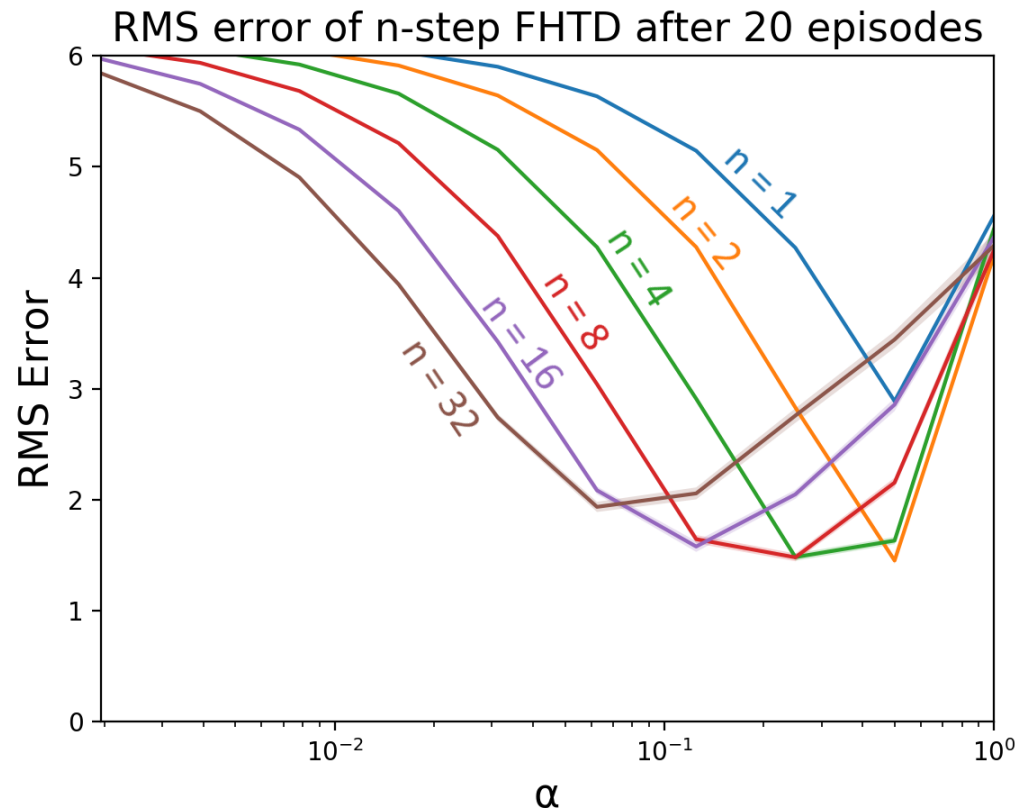


Because it's a mixture of every n -step return, you need to learn all H value functions again 😞

Multi-step FHTD prediction

In addition to the computational savings in n -step FHTD, multi-step methods address a bias-variance trade-off by adjusting the reliance on estimates being accurate

Multi-step FHTD prediction (H=32)



Used checkered grid world (De Asis et al., 2018)

Algorithm 1 Linear One-step FHTD for estimating $V^H \approx v_\pi^H$

$\mathbf{w} \leftarrow$ Array of size $(H + 1) \times m$, where $\mathbf{w}_{[0]} = [0 \text{ for } i \text{ in range}(m)]$

$s \sim p(s_0)$

$a \sim \pi(\cdot|s)$

$t \leftarrow 0$

while $t \neq t_{max}$ **do**

$s', r \sim p(s', r|s, a)$

for $h = 1, 2, 3, \dots, H$ **do**

$\delta \leftarrow r + \mathbf{w}_{[h-1]} \cdot \mathbf{x}(s') - \mathbf{w}_{[h]} \cdot \mathbf{x}(s)$

$\mathbf{w}_{[h]} \leftarrow \mathbf{w}_{[h]} + \alpha \delta \mathbf{x}(s)$

end for

$s \leftarrow s'$

$a \sim \pi(\cdot|s)$

$t \leftarrow t + 1$

end while

Algorithm 2 Linear One-step FHQ-Learning for estimating $Q^H \approx q_*^H$

$\mathbf{w} \leftarrow$ Array of size $(H + 1) \times m$, where $\mathbf{w}_{[0]} = [0 \text{ for } i \text{ in range}(m)]$

$s \sim p(s_0)$

$a \sim \mu(\cdot|s)$ (e.g. ϵ -greedy w.r.t. $Q^H(s, \cdot)$)

$t \leftarrow 0$

while $t \neq t_{max}$ **do**

$s', r \sim p(s', r|s, a)$

for $h = 1, 2, 3, \dots, H$ **do**

$\delta \leftarrow r + \max_{a'} (\mathbf{w}_{[h-1]} \cdot \mathbf{x}(s', a')) - \mathbf{w}_{[h]} \cdot \mathbf{x}(s, a)$

$\mathbf{w}_{[h]} \leftarrow \mathbf{w}_{[h]} + \alpha \delta \mathbf{x}(s, a)$

end for

$s \leftarrow s'$

$a \sim \mu(\cdot|s)$

$t \leftarrow t + 1$

end while

Algorithm 3 Linear n -step FHTD for estimating $V^H \approx v_\pi^H$

$\mathbf{w} \leftarrow$ Array of size $(\frac{H}{n} + 1) \times m$, where $\mathbf{w}_{[0]} = [0 \text{ for } i \text{ in range}(m)]$

$\mathbf{X} \leftarrow$ Array of size $n \times m$

$\mathbf{R} \leftarrow$ Array of size $n \times 1$

$s \sim p(s_0)$

$a \sim \pi(\cdot|s)$

$t \leftarrow 0$

while $t \neq t_{max}$ **do**

$\mathbf{X}_{[t \pmod n]} = \mathbf{x}(s)$

$s', r \sim p(s', r|s, a)$

$\mathbf{R}_{[t \pmod n]} = r$

if $t + 1 \geq n$ **then**

$r_{sum} \leftarrow \text{sum}(\mathbf{R})$

$\mathbf{x}_{old} \leftarrow \mathbf{X}_{[(t+1-n) \pmod n]}$

for $h_n = 1, 2, 3, \dots, \frac{H}{n}$ **do**

$\delta \leftarrow r_{sum} + \mathbf{w}_{[h_n-1]} \cdot \mathbf{x}(s') - \mathbf{w}_{[h_n]} \cdot \mathbf{x}_{old}$

$\mathbf{w}_{[h_n]} \leftarrow \mathbf{w}_{[h_n]} + \alpha \delta \mathbf{x}_{old}$

end for

end if

$s \leftarrow s'$

$a \sim \pi(\cdot|s)$

$t \leftarrow t + 1$

end while

Algorithm 4 Linear FHTD(λ) for estimating $V^H \approx v_\pi^H$

$\mathbf{w} \leftarrow$ Array of size $(H + 1) \times m$, where $\mathbf{w}_{[0]} = [0 \text{ for } i \text{ in range}(m)]$

$\mathbf{X} \leftarrow$ Array of size $H \times m$

$s \sim p(s_0)$

$a \sim \pi(\cdot|s)$

$t \leftarrow 0$

while $t \neq t_{max}$ **do**

$\mathbf{X}_{[t \pmod H]} \leftarrow \mathbf{x}(s)$

$s', r \sim p(s', r|s, a)$

for $h = 1, 2, 3, \dots, H$ **do**

$\delta^h = r + \mathbf{w}_{[h-1]} \cdot \mathbf{x}(s') - \mathbf{w}_{[h]} \cdot \mathbf{x}(s)$

for $i = 0, 1, 2, \dots, H - h$ **do**

$\mathbf{w}_{[h+i]} \leftarrow \mathbf{w}_{[h+i]} + \alpha \lambda^i \delta^h \mathbf{X}_{[(t-i) \pmod H]}$

end for

end for

$s \leftarrow s'$

$a \sim \pi(\cdot|s)$

$t \leftarrow t + 1$

end while

Algorithm 5 Linear FHTB(λ) for estimating $Q^H \approx q_{\pi^H}^H$

$\mathbf{w} \leftarrow$ Array of size $(H + 1) \times m$, where $\mathbf{w}_{[0]} = [0 \text{ for } i \text{ in range}(m)]$

$\mathbf{X} \leftarrow$ Array of size $H \times m$

$\Pi \leftarrow$ Array of size $H \times H$

$s \sim p(s_0)$

$a \sim \mu(\cdot|s)$ (e.g. ϵ -greedy w.r.t. $Q^H(s, \cdot)$)

$t \leftarrow 0$

while $t \neq t_{max}$ **do**

$\mathbf{X}_{[t \pmod H]} \leftarrow \mathbf{x}(s, a)$

$\Pi_{[t \pmod H]} \leftarrow [\pi^1(a|s), \pi^2(a|s), \dots, \pi^H(a|s)]$

$s', r \sim p(s', r|s, a)$

for $h = 1, 2, 3, \dots, H$ **do**

$\delta^h = r + \mathbf{w}_{[h-1]} \cdot \mathbb{E}_{\pi^{h-1}}[\mathbf{x}(s', \cdot)] - \mathbf{w}_{[h]} \cdot \mathbf{x}(s, a)$

$e \leftarrow 1$

for $i = 0, 1, 2, \dots, H - h$ **do**

$\mathbf{w}_{[h+i]} \leftarrow \mathbf{w}_{[h+i]} + \alpha e \delta^h \mathbf{X}_{[(t-i) \pmod H]}$

if $i \neq H - h$ **then**

$e \leftarrow e \lambda \Pi_{[(t-i) \pmod H], h+i-1}$

end if

end for

end for

$s \leftarrow s'$

$a \sim \mu(\cdot|s)$

$t \leftarrow t + 1$

end while

Algorithm 6 Linear FHQ(λ) for estimating $Q^H \approx q_*^H$

$\mathbf{w} \leftarrow$ Array of size $(H + 1) \times m$, where $\mathbf{w}_{[0]} = [0 \text{ for } i \text{ in range}(m)]$

$\mathbf{X} \leftarrow$ Array of size $H \times m$

$\Pi \leftarrow$ Array of size $H \times H$

$s \sim p(s_0)$

$a \sim \mu(\cdot|s)$ (e.g. ϵ -greedy w.r.t. $Q^H(s, \cdot)$)

$t \leftarrow 0$

while $t \neq t_{max}$ **do**

$\mathbf{X}_{[t \pmod H]} \leftarrow \mathbf{x}(s, a)$

$\Pi_{[t \pmod H]} \leftarrow [1_{a=\text{argmax}_{Q^h(s, \cdot)}} \text{ for } h \in \{1, 2, 3, \dots, H\}]$

$s', r \sim p(s', r|s, a)$

for $h = 1, 2, 3, \dots, H$ **do**

$\delta^h = r + \max_{a'} (\mathbf{w}_{[h-1]} \cdot \mathbf{x}(s', a')) - \mathbf{w}_{[h]} \cdot \mathbf{x}(s, a)$

$e \leftarrow 1$

for $i = 0, 1, 2, \dots, H - h$ **do**

$\mathbf{w}_{[h+i]} \leftarrow \mathbf{w}_{[h+i]} + \alpha e \delta^h \mathbf{X}_{[(t-i) \pmod H]}$

if $i \neq H - h$ **then**

$e \leftarrow e \lambda \Pi_{[(t-i) \pmod H], h+i-1}$

end if

end for

end for

$s \leftarrow s'$

$a \sim \mu(\cdot|s)$

$t \leftarrow t + 1$

end while
