Reinforcement Learning
Lecture 11: Recap Session

Lecturer: Prof. Dr. Mathias Niepert

Institute for Artificial Intelligence
Machine Learning and Simulation Lab

University of Stuttgart
Germany

imprs-is

July 11, 2024

Outline

Evaluation

https://evasysw.uni-stuttgart.de/evasys/online.php?p=JNGZ3

https://evasysw.uni-stuttgart.de/evasys/online.php?p=ZRDK2

Study Suggestions

Lecture to Book Chapter Mapping

The exam will cover topics covered in lectures 1-9

1. Multi-armed bandits; chapters: 2.1, 2.2, and 2.4 (+ softmax policy)
2. MDPs (definition, values function, etc.); chapters: 3.1-3.6
3. Policy improvement with dynamic programming; chapters 4.1-4.4
4. Monte-Carlo methods; chapter 5.1-5.7
5. Temporal difference methods; 6.1, 6.2, and 6.4-6.6
6. Planning and Learning; chapters: 8.1-8.4, 8.10-8.11
7. Function approximation; chapter: 9.1-9.4, 9.5.4
8. n-step bootstrapping, eligibility traces; chapters: 7.1-7.3, 12.1-12.2
9. Policy gradient methods; chapters: 13.1-13.6

General Remarks

- There will be no need to memorize or write pseudo-code (but pseudo-code typically helps to understand a method)
- You should memorize the core formulas corresponding to the various algorithms, such as the Bellman equation, MC, TD, etc.
- You should memorize backup diagrams of these formulas
- You may only use pen and scratch paper – no other materials (no textbooks, scripts, calculators, or mobiles) are allowed

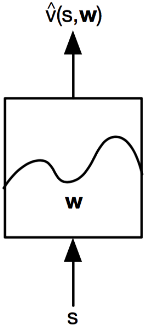
Value function approximation

Idea of value function approximation

- Parameterized functional form, with weights $\boldsymbol{w} \in \mathbb{R}^d$:

$$\hat{v}_\pi(s, \boldsymbol{w}) \approx v_\pi(s)$$

- Generally, much less weights than states $d \ll |\mathcal{S}|$
  - obvious for continuous state spaces
  - changing single weight, changes value estimate of many states
  - when one state is updated, change *generalizes* to many states
- Update $\boldsymbol{w}$ with MC or TD learning



Stochastic Gradient Descent (SGD)

- Approximate value function $\hat{v}(s, \boldsymbol{w})$
  - differentiable for all $s \in \mathcal{S}$
- Weight vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_d)^\top$
  - $\boldsymbol{w}_t$ weight vector at time $t = 0, 1, 2, \ldots$
- *Gradient* of $f(\boldsymbol{w})$: $\boldsymbol{\nabla} f(\boldsymbol{w}) = \left(\frac{\partial f(\boldsymbol{w})}{\partial w_1}, \frac{\partial f(\boldsymbol{w})}{\partial w_2}, \ldots, \frac{\partial f(\boldsymbol{w})}{\partial w_d}\right)^\top$
- Do gradient descent by sampling additive parts of the full gradient (i.e., each state consecutively)
- We can compute our update over smaller sets of inputs

Stochastic Gradient Descent (SGD) (part 2)

- When we **approximate** the gradient
- For example

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{w})$$

where $\boldsymbol{w}$ are weights.

- In machine learning

$$\mathcal{L}(\boldsymbol{w}) = -\sum_{n=1}^{N} \log p(y_n \mid \boldsymbol{x}_n, \boldsymbol{w})$$

where $\boldsymbol{x}_n \in \mathbb{R}^D$ are training *inputs* and y_n are the training *targets*

- The corresponding update

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha_t \sum_{n=1}^{N} (\nabla \mathcal{L}_n)(\boldsymbol{w}_t)$$

- Often the gradient is too difficult to compute (CPU/GPU expensive)
- **Mini-batch**: random subset
  a Large: accurate but costly
  b Small: noisy but cheaper

Stochastic Gradient Descent (SGD) (part 3)

- *Mean squared Value Error*: $\mathcal{L}(\boldsymbol{w}) = \sum_{s \in S} \mu(s) \left[v_\pi(s) - \hat{v}(s, \boldsymbol{w})\right]^2$
- Adjust $\boldsymbol{w}$ to reduce the error on sample $S_t \mapsto v_\pi(S_t)$:

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \frac{1}{2}\alpha_t (\boldsymbol{\nabla}\mathcal{L}_t)(\boldsymbol{w}_t) \\ &= \boldsymbol{w}_t - \frac{1}{2}\alpha_t \boldsymbol{\nabla} \underbrace{\left[v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)\right]^2}_{\textbf{squared sample error}} \\ &= \boldsymbol{w}_t + \alpha_t [v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)] \nabla \hat{v}(S_t, \boldsymbol{w}) \end{aligned}$$

- α_t is a step size parameter
- Why not use $\alpha = 1$, thus eliminating full error on sample?

Linear methods

- Special case where $\hat{v}(\cdot, \boldsymbol{w})$ is *linear* in the weights
- **Feature vector** $\boldsymbol{x}(s)$ represents state s:

$$\boldsymbol{x}(s) = \left[x_1(s), \quad x_2(s), \quad \dots, \quad x_d(s)\right]^\top$$

- Each component of $\boldsymbol{x}$ is a feature, examples:
  - distance of robot to landmarks
  - piece on a specific location on a chess board
- Value function is represented as a linear combination of features $\boldsymbol{x}(s)$:

$$\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

- Gradient is simply $\nabla \hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)$

Prediction

Prediction with function approximation

- We assumed the true value function $v_\pi(S_t)$ is known
- Substitute target U_t for $v_\pi(s)$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[U_t - \hat{v}(S_t, \boldsymbol{w}_t)]\nabla\hat{v}(S_t, \boldsymbol{w}_t)$$

- U_t might be a noisy or bootstrapped approximation of the true value
- **Monte Carlo:** $U_t = G_t$
- **TD(0):** $U_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}_t)$
- **TD(λ):** $U_t = G_t^\lambda$

Monte–Carlo with function approximation

- Target is unbiased by definition:

$$\mathbb{E}[U_t | S_t = s] = \mathbb{E}[G_t | S_t = s] = v_\pi(S_t)$$

- Training data:

$$\mathcal{D} = \big\{(S_1, G_1),\ (S_2, G_2),\ \ldots,\ (S_{T-1}, G_{T-1}),\ (S_T, 0)\big\}$$

- Using SGD, $\boldsymbol{w}$ is guaranteed to converge to a *local optimum*
- MC prediction exhibits local convergence with linear and non-linear function approximation
- SGD update for sample $S_t \mapsto G_t$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[G_t - \hat{v}(S_t, \boldsymbol{w})]\nabla\hat{v}(S_t, \boldsymbol{w})$$

Gradient Monte Carlo Algorithm

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

TD with function approximation

- TD-target $U_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$ is *biased* sample of the true value $v_\pi(S_t)$
- Training data:

$$\mathcal{D} = \left\{(S_1, R_2 + \gamma \hat{v}(S_2, \boldsymbol{w})),\ (S_2, R_3 + \gamma \hat{v}(S_3, \boldsymbol{w})),\ \ldots, (S_{T-1}, R_T)\right\}$$

- SGD update for sample $S_t \mapsto G_t$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})] \nabla \hat{v}(S_t, \boldsymbol{w})$$

- Linear TD(0):

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[\underbrace{R_{t+1} + \gamma \boldsymbol{w}^T \boldsymbol{x}(S_{t+1})}_{U_t\text{:TD-Target}} - \underbrace{\boldsymbol{w}^T \boldsymbol{x}(S_t)}_{\hat{v}\text{:value function}}]\boldsymbol{x}(S_t)$$

Semi-Gradient TD(0) Algorithm

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S' is terminal

Control

Action-value function approximation

- Approximate action-value function $\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$
- Linear case:

$$\hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s, a) = \sum_{i=1}^{d} w_i x_i(s, a)$$

$$\nabla \hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{x}(s, a)$$

- Minimize squared error on samples $S_t, A_t \mapsto q_\pi$: $\left[q_\pi - \hat{q}(S_t, A_t, \boldsymbol{w})\right]^2$
- Use SGD to find local minimum:

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla[q_\pi(S_t, A_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)]^2 \\ &= \boldsymbol{w}_t + \alpha[q_\pi(S_t, A_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)]\nabla\hat{q}(S_t, A_t, \boldsymbol{w})\end{aligned}$$

Control with function approximation

- Again, we must substitute target U_t for true action-value $q_\pi(s,a)$:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[U_t - \hat{q}(S_t, A_t, \boldsymbol{w}_t)]\nabla\hat{q}(S_t, A_t, \boldsymbol{w})$$

- **Monte Carlo:** $U_t = G_t$
- **One-step Sarsa:** $U_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w})$

Policy-based Reinforcement Learning

Policy–based reinforcement learning

- So far we approximated the action–value function and generated a policy from it
- Approximation of the action–value function

$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

- Generation of policy by, e.g., ϵ–greedy

$$\hat{q}(s, a, \boldsymbol{w}) \xrightarrow{\epsilon\text{-greedy}} \pi$$

- Now we directly *parameterize* the policy π

Policy optimization

- Policy optimization:

$$\pi_* = \pi(a \mid s, \boldsymbol{\theta}_*)$$

with

$$\boldsymbol{\theta}_* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

where J is some *performance measure*

- Discounted return: $G_0 = \sum_{k=0}^{T-1} \gamma^k R_{k+1}$

$$\pi_* = \arg\max_{\pi} \mathbb{E}_\pi[G_0] = \arg\max_{\pi} \mathbb{E}_\pi[v_\pi(s_0) \mid S_0 = s_0]$$

- Undiscounted return: $G_0 = \sum_{k=0}^{T-1} R_{k+1}$, i.e., $\gamma = 1$

$$\pi_* = \arg\max_{\pi} \mathbb{E}_\pi[R_0 + R_1 + \ldots + R_{T-1} \mid S_0 = s_0]$$

Parameterized policies

- Policies parameterized by parameter $\boldsymbol{\theta} \in \mathbb{R}^d$:
  - deterministic: $a = \pi(s, \boldsymbol{\theta})$
  - stochastic: $a \sim \pi(a \mid s, \boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}\}$
- Objective becomes $\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} v_{\pi_\theta}(s_0) = \max_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[G_0]$
- We can parameterize π in any way, as long as it is *differentiable* wrt to $\boldsymbol{\theta}$
- In general we require $\pi(a \mid s, \boldsymbol{\theta}) \in [0, 1]$ for all s, a
- If the action space is discrete (and not too large): **softmax policy**

$$\pi(a \mid s, \boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

  where $h(\cdot)$ is the *action preference* function
- Can this approach the deterministic policy?

Policy gradient methods

- Problem:

$$\pi_* = \pi(a \mid s, \boldsymbol{\theta}_*)$$

with

$$\boldsymbol{\theta}_* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[G_0]$$

Intuition: collect a bunch of trajectories, and ...

1. Make the good (high return) trajectories more probable by
2. Making the actions of these good trajectories more probable by
3. Pushing the policy toward generating these good actions

- Policy gradient methods:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \widehat{\nabla J(\boldsymbol{\theta})}$$

  - where $\nabla J(\boldsymbol{\theta})$ is the *policy gradient*:

$$\nabla J(\boldsymbol{\theta}) = \left(\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0}, \ldots, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_d} \right)^T$$

Score function

- We now compute the policy gradient analytically
- Assume policy $\pi_{\boldsymbol{\theta}}$ is differentiable whenever it is non-zero
- We have the following useful identity

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a \mid s) &= \pi_{\boldsymbol{\theta}}(a \mid s) \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_{\boldsymbol{\theta}}(a \mid s)} \\ &= \pi_{\boldsymbol{\theta}}(a \mid s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)\end{aligned}$$

- The **score function** is $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)$

Gradient of expectation → expectation of gradient

- Consider $\mathbb{E}_{x \sim p(x|\boldsymbol{\theta})}[f(x)]$ for some function f
- We want to compute the gradient wrt $\boldsymbol{\theta}$

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathbb{E}_x[f(x)] &= \nabla_{\boldsymbol{\theta}} \int f(x) p(x \mid \boldsymbol{\theta}) dx \\
&= \int f(x) \nabla_{\boldsymbol{\theta}} p(x \mid \boldsymbol{\theta}) dx \\
&= \int f(x) \frac{\nabla_{\boldsymbol{\theta}} p(x \mid \boldsymbol{\theta})}{p(x \mid \boldsymbol{\theta})} p(x \mid \boldsymbol{\theta}) dx \\
&= \int \left[f(x) \nabla_{\boldsymbol{\theta}} \log p(x \mid \boldsymbol{\theta}) \right] p(x \mid \boldsymbol{\theta}) dx \\
&= \mathbb{E}_x[f(x) \underbrace{\nabla_{\boldsymbol{\theta}} \log p(x \mid \boldsymbol{\theta})}_{\text{score}}]
\end{aligned}
$$

Score: Softmax Policy

- Weight actions using linear combination of features $h(s, a) = \phi(s, a)^\top \boldsymbol{\theta}$
- Probability of action is proportional to exponentiated weight

$$\pi_{\boldsymbol{\theta}} = \frac{\exp\left(\phi(s, a)^\top \boldsymbol{\theta}\right)}{\sum_a \exp(\phi(s, a)^\top \boldsymbol{\theta})}$$

- The score function is

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) = \phi(s, a) - \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\phi(s, \cdot)]$$

Score: Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\beta(s) = \phi(s)^\top \boldsymbol{\theta}$
- Variance may be fixed σ^2, or can also parametrized
- Policy is Gaussian, $a \sim \mathcal{N}(\beta(s), \sigma^2)$
- The score function is

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) = \frac{(a - \beta(s))\, \phi(s)}{\sigma^2}$$

One-Step MDPs

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim \mu(s)$
  - In general, $\mu(s)$ is on-policy distribution, meaning the probability that we are in state s
  - Terminating after one time-step (taking action a) with reward $r = R_{s,a}$
- Use score function to compute the policy gradient

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[r] \\
&= \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s) R_{s,a} \\
\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s) \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) R_{s,a} \\
&= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[r \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)]
\end{aligned}
$$

Policy Gradient

- $\Pr(s_0 \to s, k, \pi)$ is the probability to transition from state s_0 to s in k steps under policy π
- $\eta(s) = \sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi)$ is the number of average time steps spent in a single episode
- $\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$ is probability of being in state s

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\propto \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s) \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) q_{\pi}(s, a) \\ &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[q_{\pi}(S_t, A_t) \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)] \end{aligned}$$

Policy Gradient Theorem

- The policy gradient theorem generalises the previous derivation to multi-step MDPs
- Replaces instantaneous reward r with long-term value $q_\pi(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem (Policy-gradient)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[q_\pi(s, a) \nabla_{\boldsymbol{\theta}} \log \pi(a \mid s, \boldsymbol{\theta})\right]$$

Update rule: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})$

Why γ^t?

- $\Pr(s_0 \to s, k, \pi)$ is the probability to transition from state s_0 to s in k steps under policy π
- With $\gamma < 1$, we need to use $\eta_\gamma(s) = \sum_{k=0}^{\infty} \gamma^k \Pr(s_0 \to s, k, \pi)$
- $\mu_\gamma(s) = \frac{\eta_\gamma(s)}{\sum_{s'} \eta_\gamma(s')}$

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\propto \sum_{s \in \mathcal{S}} \mu_\gamma(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) q_\pi(s, a) \\
&= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\gamma^t q_\pi(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)]
\end{aligned}
$$

Policy Gradient Methods

Monte–Carlo policy gradient (REINFORCE)

- Update parameters $\boldsymbol{\theta}$ by stochastic gradient ascent
  - using the policy gradient theorem
  - using return G_t as an unbiased sample of $q_\pi(S_t, A_t)$

```
Initialize a differentiable policy parameterization π(a|s, θ)
Initialize θ ∈ ℝ^{d'}
for all episodes do
    Generate an episode τ = (S_0, A_0, R_1, ..., S_{T−1}, A_{T−1}, R_T, S_T)
following π(.|., θ)
    for t = 0, 1, ..., T − 1 do
        G_t ← Σ_{k=t+1}^{T} γ^{k−t−1} R_k              // return at time t
        θ ← θ + αγ^t G_t ∇_θ log π(A_t | S_t, θ)       // update rule
    end for
end for
```

REINFORCE with baseline

- Monte–Carlo policy gradient still suffers from high variance

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s) \left(q_\pi(s,a) - b(s)\right) \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)$$

Theorem (Policy-gradient with baseline)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\left(q_\pi(s,a) - b(s)\right) \nabla_{\boldsymbol{\theta}} \log \pi(a \mid s, \boldsymbol{\theta})\right]$$

Update rule: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t \left(G_t - b(S_t)\right) \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})$

REINFORCE with baseline

- We use an approximation of the value function $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$ as a *baseline*
- Policy parameters: $\boldsymbol{\theta}$
- Value estimator parameters: $\boldsymbol{w}$
- Update rule:

$$\begin{aligned}
\delta &\leftarrow G - \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \gamma^t \delta \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})
\end{aligned}$$

- Interpretation:
  - $\uparrow$ log–prob of action A_t proportionally to how much G_t is better than expected
  - baseline accounts for and removes the effect of past actions

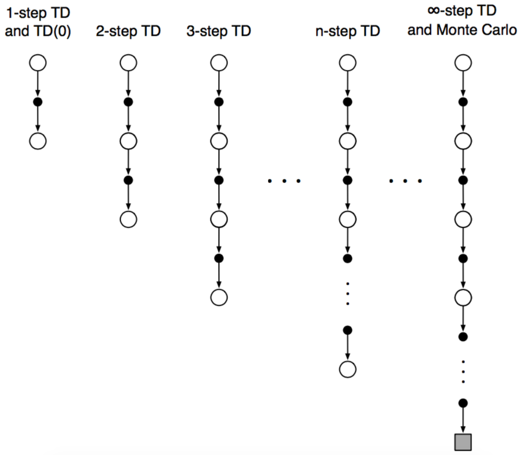
Actor-critic vs. baseline

- Delivers trade off between *variance reduction* of policy gradients with *bias introduction* from value function methods
- **Critic:** updates value–function parameters $\boldsymbol{w}$
- **Actor:** updates policy parameters $\boldsymbol{\theta}$ using critic
- REINFORCE with baseline uses value–function as *baseline* not as a *critic*
  - not used for *bootstrapping*
- One–step actor-critic update:

$$
\begin{aligned}
\delta &\leftarrow R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \gamma^t \delta \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \log \pi(A_t \mid S_t, \boldsymbol{\theta})
\end{aligned}
$$

n-step Methods and Eligibility Traces

n-step TD prediction



n-step returns

- Monte Carlo:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$$

- TD:

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$$

- 2-step return:

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

- n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Error-reduction property

- *Error reduction property* of n-step returns

$$\underbrace{\max_s \left|\mathbb{E}_\pi[G_{t:t+n} \mid S_t = s] - v_\pi(s)\right|}_{\text{Maximum error using } n\text{-step return}} \leq \underbrace{\gamma^n \max_s \left|V_{t+n-1}(s) - v_\pi(s)\right|}_{\text{Maximum error using } V}$$

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

- Using above, we can show that n-step methods converge
- Generalization of 1-step:

$$\max_s \left|\mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s] - v_\pi(s)\right| \leq \gamma \max_s \left|V(s) - v_\pi(s)\right|$$

n-step TD

- n-step return

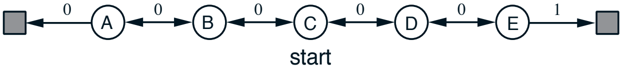
$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

- *Not available* until time $t+n$
- Natural algorithm is to wait until time $t+n$
- n-**step TD** update:

$$V_{\underbrace{t+n}_{\text{next step}}}(S_t) = V_{\underbrace{t+n-1}_{\text{previous step}}}(S_t) + \alpha\left[G_{t:t+n} - V_{\underbrace{t+n-1}_{\text{previous step}}}(S_t)\right]$$

Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
for all episode **do**
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    repeat for $t = 0, 1, 2, \ldots$
        if $t < T$ **then**
            Take an action according to $\pi(\cdot \mid S_t)$
            Observe and store next reward R_{t+1} and state S_{t+1}
            if S_{t+1} is terminal **then** $T \leftarrow t + 1$
        end if
        $\tau \leftarrow t - n + 1$ ▷ τ is the time whose state's estimate is updated
        if $\tau \geq 0$ **then**
            $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
            if $\tau + n < T$ **then** $G \leftarrow G + \gamma^n V(S_{\tau+n})$
            $V(S_\tau) \leftarrow V(S_\tau) + \alpha[G - V(S_\tau)]$
        end if
    until $\tau = T - 1$
end for

Random walk example



- Start with $V(s) = 0.5$ for all s
- Suppose the first episode progressed directly from C to the right, through D and E
- How does 2-step TD work here?
- How about 3-step TD?
- n-**step TD** update:

$$V_{\underbrace{t+n}_{\text{next step}}}(S_t) = V_{\underbrace{t+n-1}_{\text{previous step}}}(S_t) + \alpha \Big[G_{t:t+n} - V_{\underbrace{t+n-1}_{\text{previous step}}}(S_t) \Big]$$

n-step Sarsa

- Action–value of n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

- n-step Sarsa:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \Big[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \Big]$$

- n-step Expected Sarsa:
  - same update
  - *slightly* different n-step return

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a \mid S_{t+n}) Q_{t+n-1}(S_{t+n}, a)$$

Initialize action-value function parameterization $\hat{q}$
for all episode **do**
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot \mid S_0)$ or ϵ-greedy wrt $\hat{q}$
  $T \leftarrow \infty$
  repeat for $t = 0, 1, 2, \dots$
    if $t < T$ **then**
      Take an action A_t
      Observe and store next reward R_{t+1} and state S_{t+1}
      if S_{t+1} is terminal **then** $T \leftarrow t + 1$
      else Select and store an action $A_{t+1} \sim \pi(\cdot \mid S_{t+1})$ or ϵ-greedy wrt $\hat{q}$
    end if
    $\tau \leftarrow t - n + 1$ ▷ τ is the time whose state's estimate is updated
    if $\tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      if $\tau + n < T$ **then** $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n})$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[G - \hat{q}(S_\tau, A_\tau, \boldsymbol{w})] \boldsymbol{\nabla} \hat{q}(S_\tau, A_\tau, \boldsymbol{w})$
    end if
  until $\tau = T - 1$
end for

n-step off–policy learning

- Recall the importance sampling ratio:

$$\rho_{t:h} = \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k \mid S_k)}{\mu(A_k \mid S_k)}$$

- Off–policy methods weight updates by this ratio
- Off–policy n-step TD:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \Big[G_{t:t+n} - V_{t+n-1}(S_t) \Big]$$

n-step off–policy learning (part 2)

- Recall the importance sampling ratio:

$$\rho_{t:h} = \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k \mid S_k)}{\mu(A_k \mid S_k)}$$

- Off–policy n-step Sarsa:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} \Big[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \Big]$$

- Off–policy n-step Expected Sarsa:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n-1} \Big[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \Big] \text{ , with}$$

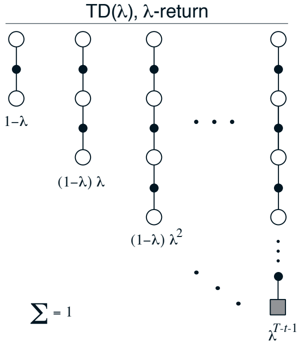
$$G_{t:t+n} = R_{t+1} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a \mid s) Q_{t+n-1}(s, a)$$

Eligibility Traces

λ-return

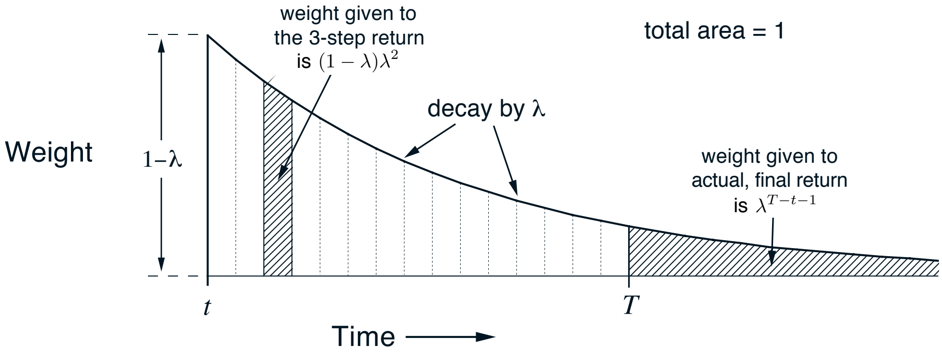
- The λ-return G_t^λ combines all n-step returns (weighted averaging):

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}$$



λ-return weighting function

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{T-t-1}G_t$$



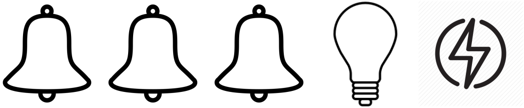
λ-return weighting function (part 2)

- General weighting function:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

- For $\lambda = 1$: $G_t^\lambda = G_t$ (Monte Carlo)
- For $\lambda = 0$: $G_t^\lambda = G_{t:t+1}$ (1-step TD)

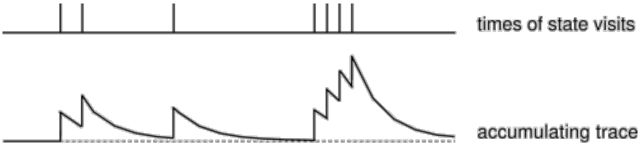
Eligibility traces



Credit assignment problem:

- **Frequency:** assign credit to most frequent states
- **Recency:** assign credit to recent states

$$\forall s : e(s) \leftarrow \gamma\lambda e(s)$$
$$e(S_t) \leftarrow e(S_t) + 1$$



TD(λ) and TD(0)

- When $\lambda = 0$:

$$e(s) = \begin{cases} 1 & \text{for} \quad s = S_t \\ 0 & \text{else} \end{cases}$$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$\forall s : V(s) \leftarrow V(s) + \alpha \delta_t e_t(s)$$

- Same as TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

- What if $\lambda = 1$? Monte-Carlo

TD(λ) with function approximation

- Eligibility trace vector $\boldsymbol{e}$ keeps track which components have contributed to recent state evaluations
- Indicate the *eligibility* of each component for undergoing learning

$$\begin{aligned}
\delta &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{e} &\leftarrow \gamma \lambda \boldsymbol{e} + \nabla \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha \delta \boldsymbol{e}
\end{aligned}$$

- Update weight vector proportional to scalar TD error and eligibility trace vector