LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME UND DATA MINING

Lecture Notes for

# Deep Learning and Artificial Intelligence

Winter Term 2018/2019

## Model-Free Reinforcement Learning



DBS

# Why model-free Reinforcement Learning?

situations where MDPs are difficult to apply:

- we do not know the exact mechanics of the environment we are acting in
- computing transition functions might be to complicated

$\Rightarrow$ model-free approaches learn while acting without explicitly modelling the environment

$\Rightarrow$ requires an environment reacting to the agent

# in this lecture

- model free approaches on discrete state spaces
  - ⇒ we still learn an action for every state
  - ⇒ large state spaces still yield problems
- model free prediction based on:
  - learning from complete episodes (Monte Carlo Learning)
  - learning based on bootstrapping (Temporal Difference Learning)
  - combinations (TD($\lambda$) approach)
- model free control:
  - exploration vs. exploitation
  - on-policy learning (SARSA)
  - off-policy learning (Q-Learning)

# Model free Prediction

Given policy π, we want to predict $U^{\pi}(s)\ \forall\, s \in S$.

Without a model we cannot compute the expectation directly. We have to sample from the environment.

Two ways to do this:

- sample complete episodes and afterwards update the empirical mean of all states (Monte-Carlo Learning)

- sample one step and update the estimate by the direct reward and the current estimate of the next step (temporal difference learning)

# Monte-Carlo Learning

- learn directly on complete episodes $S_1, A_1, R_1, S_2, .. O_t, R_t$
  $\Rightarrow$ allows for seeing all future rewards
- does not need a model/MDP transitions or the exact distribution of rewards
- no estimate about the future is involved (no bootstraping)
- estimate the expected reward by the empirical mean of future rewards when following policy $\pi$
- BUT: Can only be applied to episodic problems (episodes must terminate to be complete)

# Monte-Carlo Policy Evaluation

Given policy π, we want to predict $U^{\pi}(s)\ \forall\ s \in S$ and a set of episodes of experience x ∈ X:

$$x = s_1, r_1, a_1, s_2, r_2, a_2, \ldots, a_{t-1}, s_t, r_t \sim \pi$$

remember: $G_t(x) = \sum_{i=t}^{l} \gamma^{i-t} r_{i+1}^{x}\ with\ 0 < \gamma \leq 1$

$$and\ U^{\pi}(s_i) = \mathbb{E}[G_t | s_t = s_i, \pi]$$

⇒ estimate the utility Monte-Carlo Learning uses the empirical mean over X

# Monte-Carlo Policy Evaluation

for a known policy π and a set of complete sample episodes X following π:

- let X(s) be the set of (sub-)episodes starting with s
- to estimate utility $U^{\pi}(s)$ average over the expected reward:

$$\mathrm{U}(s) = \sum_{x \in X(s)} \frac{\sum_{i=1}^{l} \gamma^i r_i}{\lfloor X(s) \rfloor}$$

- if |X(s)| is sufficiently large for all $s \in S$:

  $$U(s) \rightarrow U^{\pi}(s)$$

  (c.f. law of large numbers)

# First Visit and Every Visit MCPE

sometimes an episode x might visit state s more than once:

- ***First Visit Monte-Carlo Policy Evaluation*** only considers the first time s is visited in x

  => in case of costs pessimistic

  => in case of rewards optimistic

- ***Every Visit Monte-Carlo Policy Evaluation*** considers every time s is visited in x

  => considers postfixes of the same episode multiple time

# Incremental Monte-Carlo updates

- when training on an environment, we usually sample until the result is stable => incremental training
- compute incremental mean:

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) = \frac{1}{k}(x_k + (k-1)\mu_{k-1})$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

- if the environment is non-stationary, limit weight of old episodes:

$$\mathrm{U}(s_t) \leftarrow \mathrm{U}(s_t) + \alpha\big(G_t - \mathrm{U}(s_t)\big)$$

# Temporal Difference Learning

**problem**: Can we still learn if episodes are incomplete?
- the later part of $\sum_{i=1}^{l} \gamma^i r_i$ is missing
- in the extreme case, we just have 1 Step: $s_t$, a, r, $s_{t+1}$

=> ***Temporal Difference Learning***
- idea similar to incremental Monte-Carlo learning:

$$\mathrm{U}(s_t) \leftarrow \mathrm{U}(s_t) + \alpha\big(G_t - \mathrm{U}(s_t)\big)$$

- Policy Evaluation with Temporal Difference (TD) Learning:

$$\mathrm{U}(s_t) \leftarrow \mathrm{U}(s_t) + \alpha\big(R(s_{t+1}) + \gamma \mathrm{U}(s_{t+1}) - \mathrm{U}(s_t)\big)$$

- TD target: $R(s_{t+1}) + \gamma \mathrm{U}(s_{t+1})$
- TD error: $R(s_{t+1}) + \gamma \mathrm{U}(s_{t+1}) - \mathrm{U}(s_t)$
- each step estimates the mean utility incrementally

# Properties of TD

- TD can learn online after every step
  - ⇒ TD does not have to wait until the episode ended to update U(s)
  - ⇒ TD works in continuing (non-terminating environments

- TD learns based on an estimate of the future development for the next state
  - ⇒ initial estimate influence convergence
  - ⇒ may add Bias during training

# Bias and Variance

To further compare MC and TD learning, we have to examine the properties of the way U($s_t$) is computed.

- $G_t(x) = \sum_{i=t}^{l} \gamma^{i-t} r_{i+1}^{x}$ is an unbiased estimate of $U_\pi(S_t)$
- the true TD target : $R(s_{t+1}) + \gamma U_\pi(s_{t+1})$ is also an unbiased estimate of $U_\pi(S_t)$
- given the TD estimate $\widehat{U}_\pi(S_t)$ of the $U_\pi(S_t)$, $R(s_{t+1}) + \gamma \widehat{U}_\pi(s_{t+1})$ is biased

$\Rightarrow$ TD has a smaller variance than $G_t$:

- $G_t$ depends on many random actions, transitions, rewards
- TD only depends on one action, transition and reward

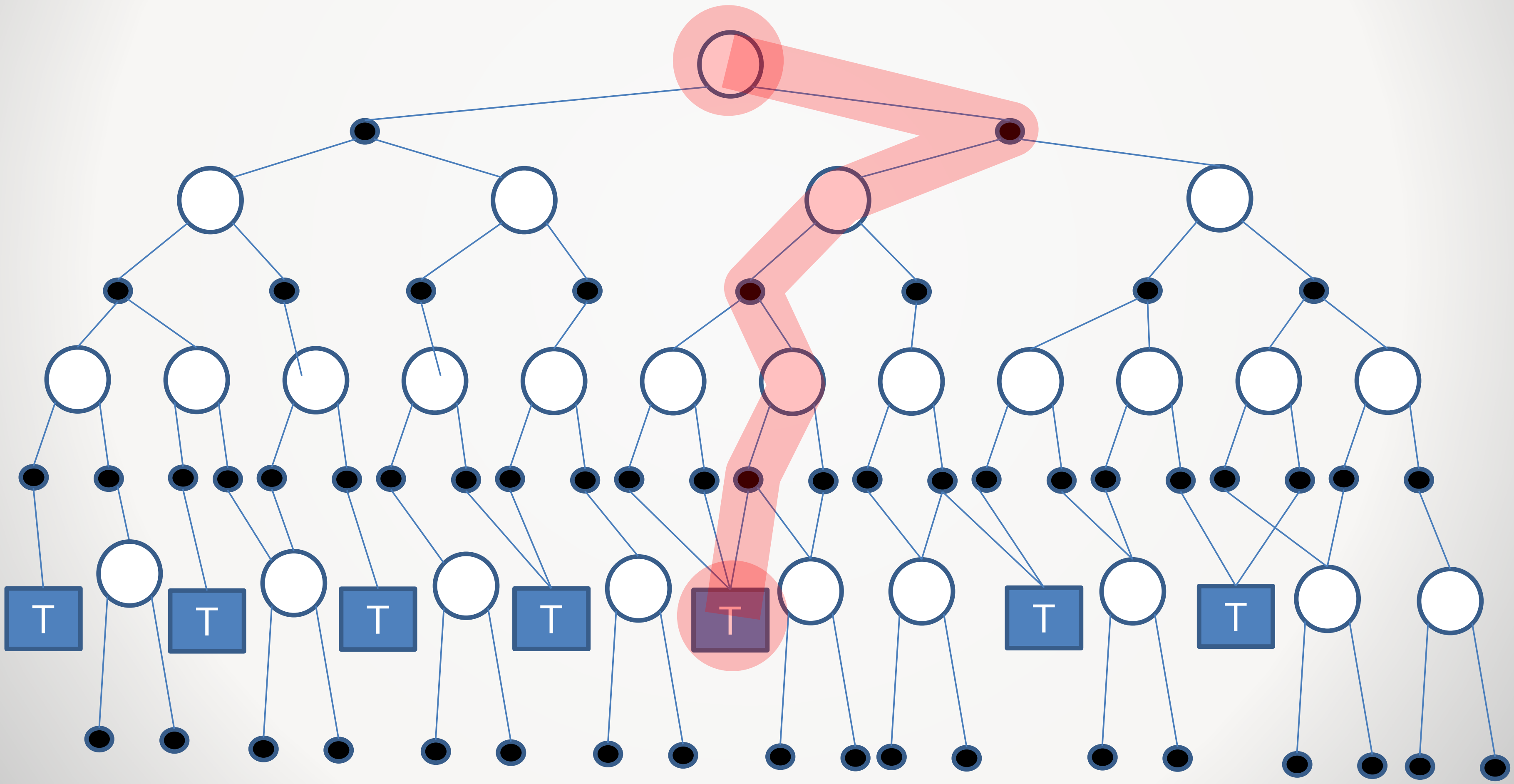
# TD vs. MC Summary

- MC has high variance but no bias
  - stable convergence against $U\pi(s)$
  - not very sensitive to initial estimate
  - simple to use
  - might suffer from insufficient samples

- TD has low variance, but is biased
  - usually faster convergence than MC
  - sensitive to initial models
  - TD(0) still converges to $U\pi(s)$ in most cases (function approximations might cause problems)
  - copes  well with limited samples due to exploiting the Markov property

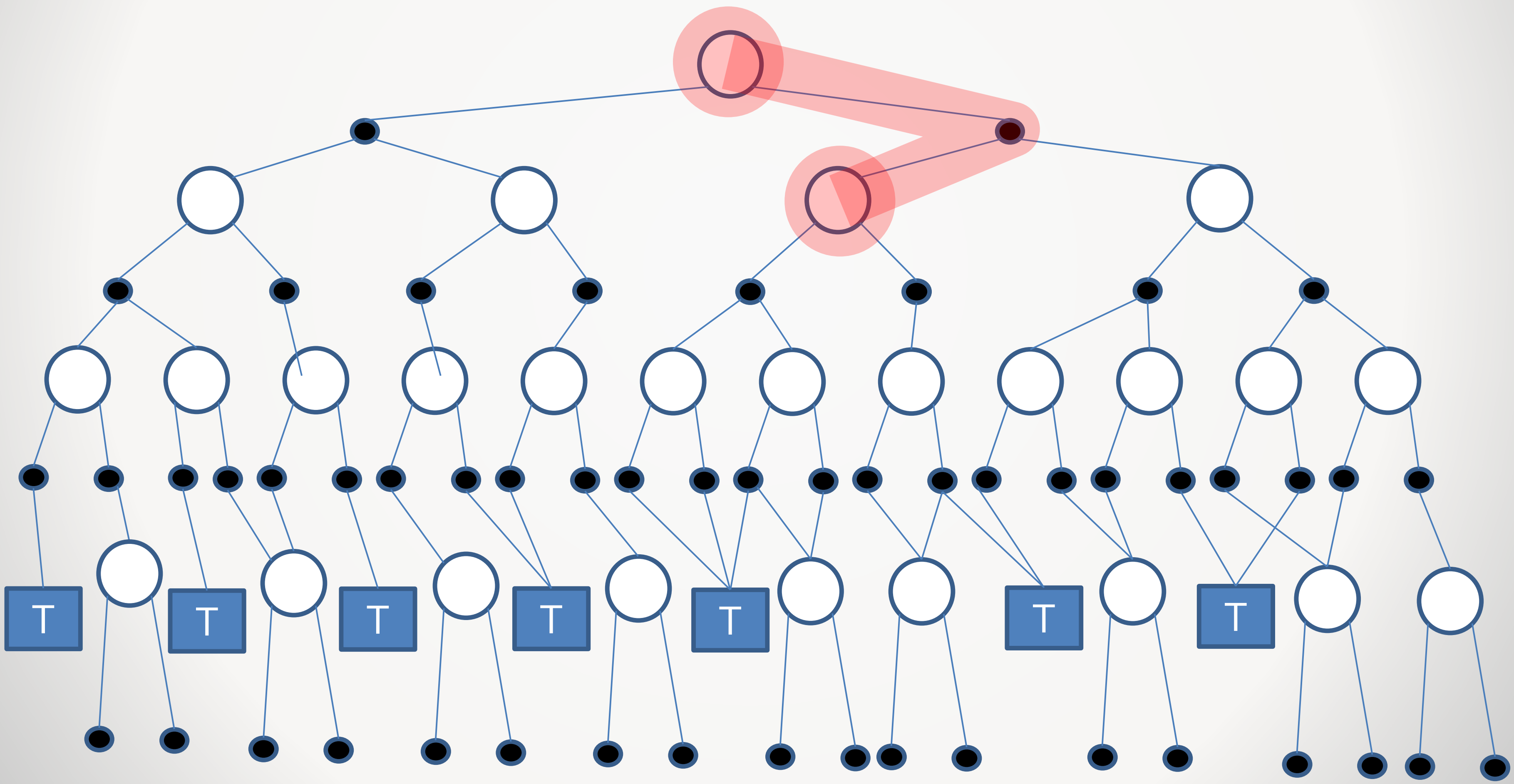
# Recap Backup Strategies

Monte Carlo Backup: $U(S_t) \leftarrow U(S_t) + \alpha\big(G_t - U(S_t)\big)$



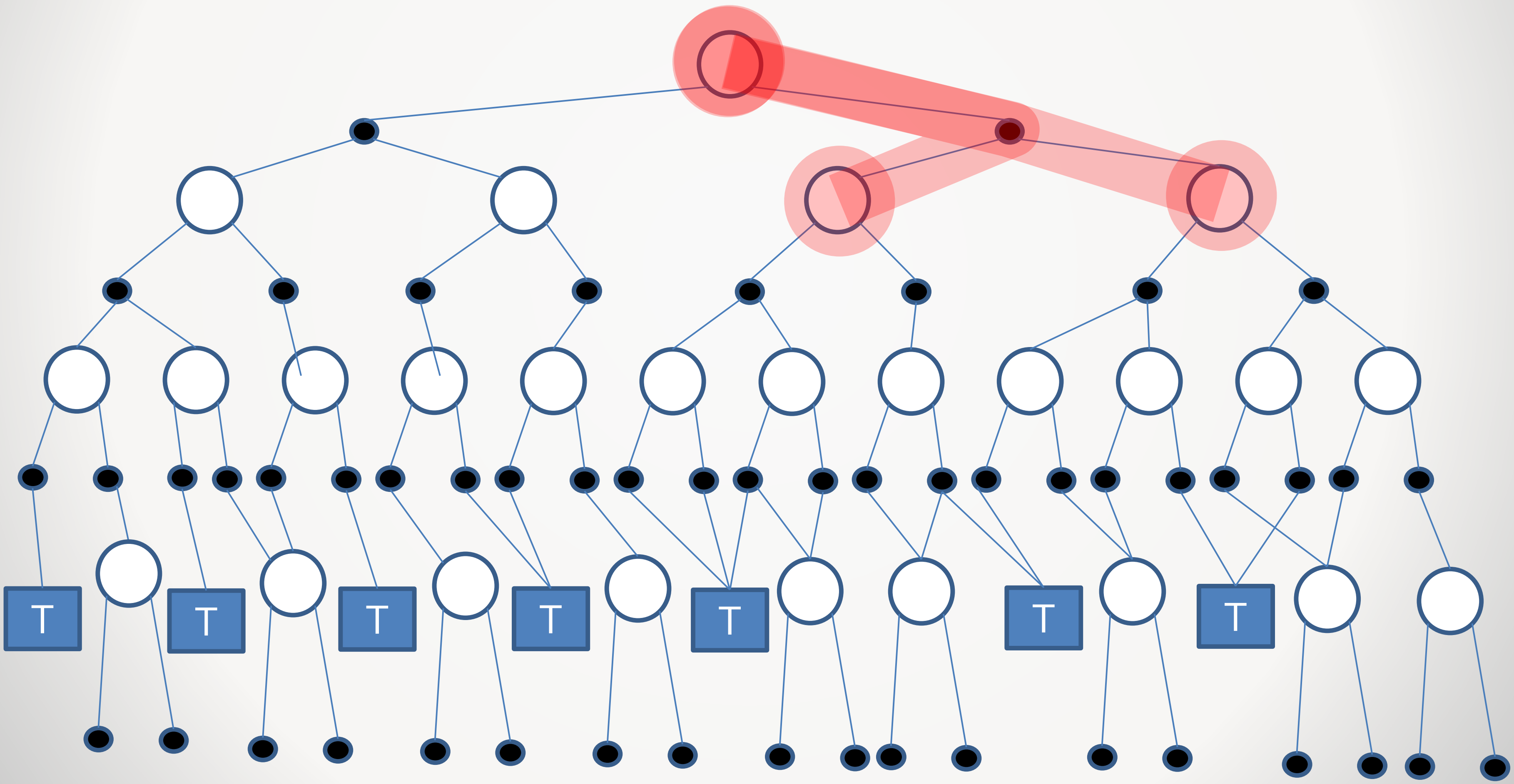
# Recap Backup Strategies

Temporal Difference Backup: $U(S_t) \leftarrow U(S_t) + \alpha\big(R(s_{t+1}) + \gamma \mathrm{U}(s_{t+1}) - \mathrm{U}(s_t)\big)$



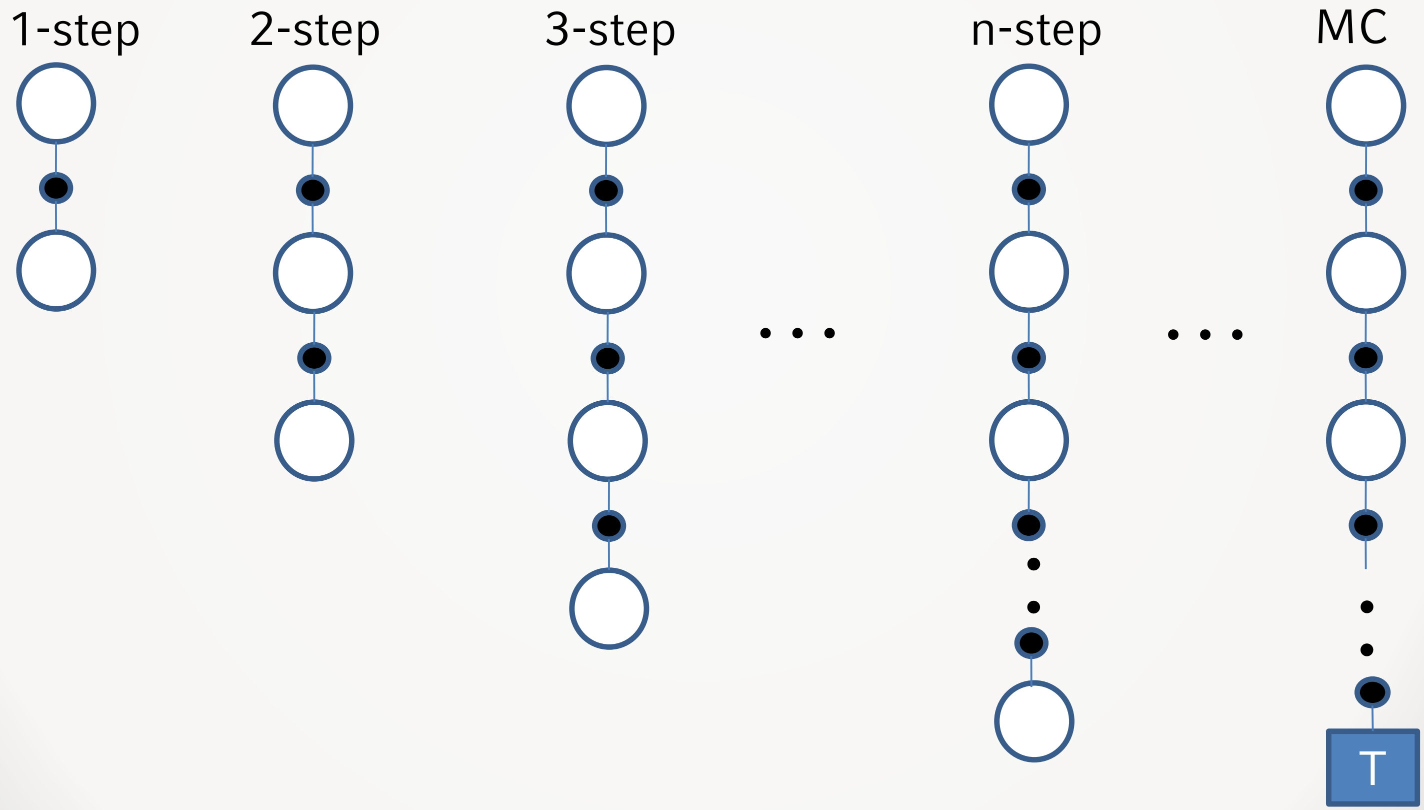
# Recap Backup Strategies

Dynamic Programming Backup: $U(S_t) \leftarrow \mathbb{E}[R(s_{t+1}) + \gamma \mathrm{U}(s_{t+1})]$



# n-step Prediction

**idea**: TD looks 1 step ahead and MC looks until the end of the episode. Why not combine and look n steps ahead?



# n-step Return

- return for n = 1,2,..,∞:

  n=1 : $G_t^{(1)} = R_{t+1} + \gamma U(S_{t+1})$ (TD)

  n=2 : $G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 U(S_{t+2})$

  $\vdots$

  n=∞ : $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ (MC)

- general n-step return:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n U(S_{t+n})$$

- n-step temporal difference learning:

$$U(S_t) \leftarrow U(S_t) + \alpha\left(G_t^{(n)} - \mathrm{U}(s_t)\right)$$

# properties of n-step TD learning

**Pro**:

- less bias than TD learning and converges faster than MC

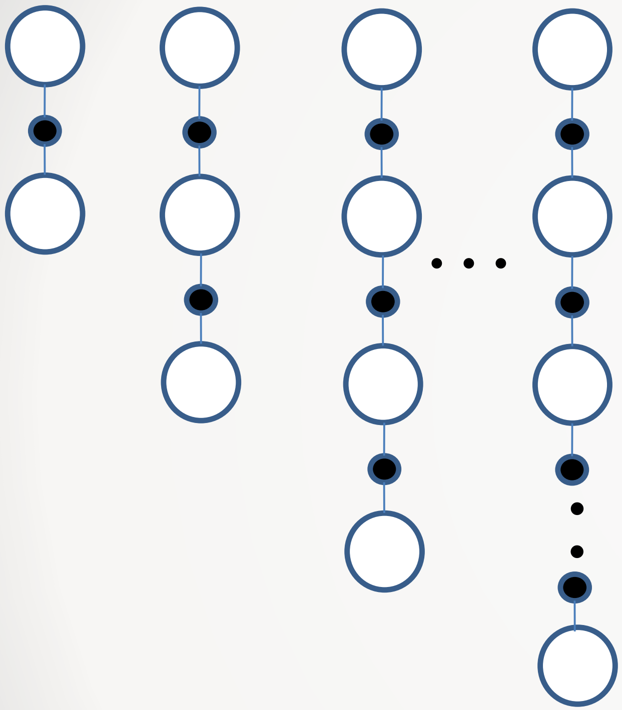
**Con**:

- episodes might strongly vary in length
- choice of n might influence convergence speed

$\Rightarrow$ combine multiple value for n

$\Rightarrow$ average over all values for n

# TD(λ) and λ Returns



1-λ (1-λ) λ (1-λ) $\lambda^2$ $\lambda^{T-t-1}$

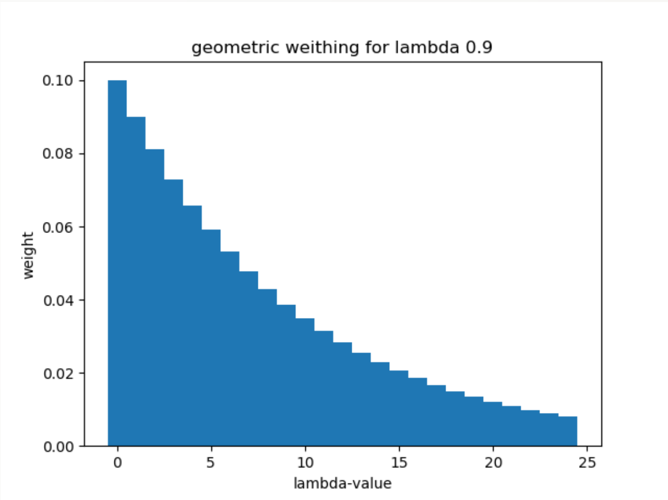
- λ-return $G_t^{\lambda}$ combines all n-step returns $G_t^{(n)}$
- Using weight (1-λ) $\lambda^{n-1}$ :

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(n)}$$

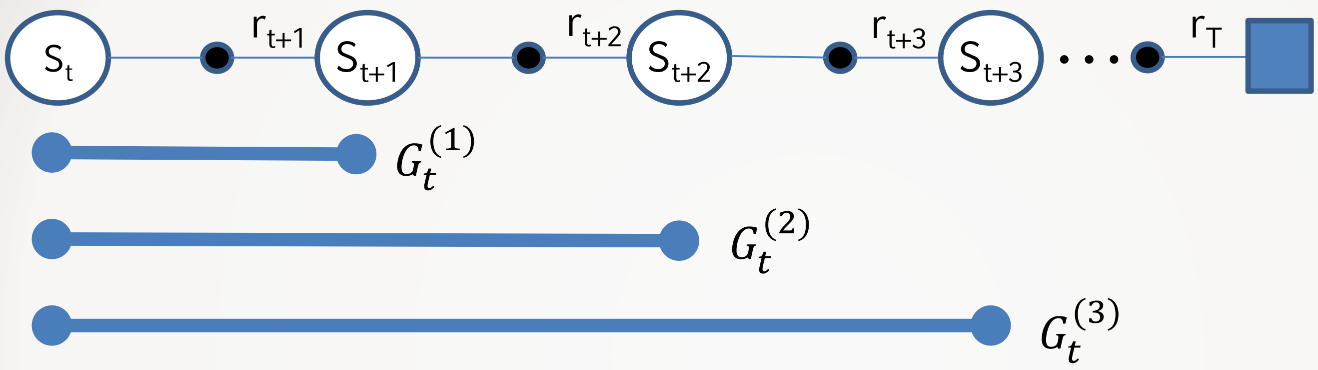
(geometric weighting)

- Forward-view TD(λ):

$$U(S_t) \leftarrow U(S_t) + \alpha\left(G_t^{\lambda} - \mathrm{U}(s_t)\right)$$



# Forward TD($\lambda$)




- update utility towards λ-return $G_t^\lambda$
- forward-view looks into the future to compute $G_t^\lambda$
- suffers from the same problems as MC

  => has to compute complete episodes

# Backward View

To practically apply *TD(λ)*, we want a method which does not have to wait until the end of the episode.

⇒ look backward and update return of states visited so far

Idea:

- in the forward approach, we consider each state and update $U(S_t)$ for the remaining episode
- in the backward approach, we update the $U(S_{t-i})$ for previous states $S_i$ after observing reward $R_t$
  - each $R_t$ influences the utility of all previous states $S_i$
  - part of the TD error for $S_i$ relates to the later $R_t$

⇒ update the $U(S_{t-i})$ with a part of the recent TD error:

TD-error: $\delta_t = R_{t+1} + \gamma U(S_{t+1}) - U(S_t)$

TD-update: $U(s) \leftarrow U(s) + \alpha\delta_t E_t(s)$

where $E_t(s)$ is measure describing the eligibility of $\delta_t$ for state s.

# Eligibility Traces

- We need a measure to describe how strong the current TD error influences the utility of state s U(s)
- general heuristics: frequency and recency
  - The more recent a state s was visited, the more U(s) is influenced by $\delta_t$
  - The more often a state s, the more U(s) is influenced by $\delta_t$

$\Rightarrow$ eligibility traces combine both heuristics

$$E_o(s) = 0$$

$$E_t(s) = \underbrace{\gamma\lambda E_{t-1}(s)}_{\text{decayed old egilibility}} + \underbrace{1(S_t = s)}_{\text{inc when observing state s by 1}}$$

# TD(λ) and TD(0)

For λ=0, $E_t(s) = 1(S_t = s)$:

$$U(s) \leftarrow U(s) + \alpha\delta_t E_t(s)$$
$$= U(s) + \alpha\delta_t$$
$$= U(s) + \alpha\big(R_{t+1} + \gamma U(S_{t+1}) - U(s)\big)$$

- λ=0 means only the previous state is updated
- this corresponds to the original 1-step TD learning
- In the forward view:

$$G_t^0 = (1-0)\sum_{n=1}^{\infty} 0^{n-1}\, G_t^{(n)} = (1-0)\sum_{n=1}^{\infty} 0^{n-1}\left(U(S_{t+n}) + \sum_{j=1}^{n} R_{t+j}\right)$$
$$= 0^0\big(U(S_{t+1}) + \textstyle\sum_{j=1}^{1} R_{t+j}\big) + \textstyle\sum_{n=2}^{\infty} 0^{n-1}\,\big(U(S_{t+n}) + \textstyle\sum_{j=1}^{n} R_{t+j}\big)$$
$$= U(S_{t+1}) + R_{t+1}$$

# TD(λ) and TD(0)

- for λ=1 (credit is kept until end of the episode)
- in an episodic environment (finite episodes)

⇒ over the course of an episode, the total updates w.r.t. TD(1) are the same as the total updates for MC

***Theorem***: The sum of offline updates is identical for forward-view and backward-view TD(λ):

$$\sum_{t=1}^{T} \alpha \delta_t E_t = \sum_{t=1}^{T} \alpha \left( G_t^{\lambda} - U(S_t) \right) 1(S_t = s)$$

# TD(1) and MC Learning

- TD(1) is roughly equivalent to every-visit Monte-Carlo
- error is accumulated online, step-by-step
- If the utility is only updated offline at the end of the episode then the total update is exactly the same as MC

# Model Free Control

***so far***: We can evaluate policy $\pi$ based on observing it in an Environment. But what is a good policy?

Example tasks for model-free control:

- control robots
- play games
- control automatic systems
- ..

When MDP is unknown or too big to use:

=> sample experience from an environment and employ model-free control

# Policy Optimization

**Idea**: adapt Policy Iteration
(evaluate policy and update greedily)

- greedy policy update of U(s) requires MDP:
$$\pi'(s) = R(s) + argmax_{a \in A(s)} P(s'|s,a) U(s')$$
- Q-Value Q(s,a): If we choose action a in state s what is the expected reward?
=> We do not need to know where action a will take us!
- Improving Q(s,a) is model free:
$$\pi'(s) = argmax_{a \in A(s)} Q(s,a)$$
- Adapt the idea of Policy Iteration:
  - Start with a default policy
  - evaluate policy (previous slide)
  - update policy: e.g. with greedy strategy

# Samples and Policy Updates

***Problem:*** After updating a policy, we need enough samples following the policy.

- real observed episodes usually do not cover enough policies (episodic samples are policy dependent)

$$s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, s_3 \ldots, s_l, a_l, r_l, s_{l+1}$$

- we need to dynamically sample from an environment:
  - measure reaction of physical world (e.g. robotics..)
  - build simulations which mimic the physical world
  - in Games: let the agent play and learn !!!
- We need a strategy for sampling these (s,a) pairs.
- s is often determined by the environment as result of the last action.

# Learning on a Queryable Environment

- we can generate as much samples as possible
- environment might be non-deterministic:
  - same state s and action a
    => different outcomes s' and R(s')
  - multiple samples for the same (s,a) might be necessary

- How to sample over the state-action space?
  - **exploit**: If we find a good action keep it and improve the estimate of Q(s,a). Usually, it's a waste of time to optimize Q(s,a) for bad actions.

  - **explore**: Select unknown or undersampled actions
    - a low Q(s,a) need not mean that the option is bad, maybe it is just underexplored.
    - try out new things might lead to a even better solution

# ε-Greedy Exploration

- makes sure that sampling considers new actions
- when sampling:
  - with probability 1-ε choose greedy action
  - with probability ε chose random action
- Sampling policy:

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{m} + (1-\varepsilon) & if\ a = argmax_{a \in A(s)} Q(s,a) \\ \frac{\varepsilon}{m} & otherwise \end{cases}$$

- achieves that Q-values improve and guarantees that all actions are explored if optimized long enough

# ε-Greedy improvement

**Theorem:** For any ε-greedy policy π, the ε-greedy policy π′ with respect to $q_\pi$ is an improvement, i.e., $v_{\pi'}(s) \geq v_\pi(s)$.

Proof:

$$q_\pi\big(s, \pi'(s)\big) = \sum_{a \in A} \pi'(a|s)\, q_\pi(s,a)$$

$$= \frac{\varepsilon}{m} \sum_{a \in A} q_\pi(s,a) + (1-\varepsilon) \max_{a \in A} q_\pi(s,a)$$

$$\geq \frac{\varepsilon}{m} \sum_{a \in A} q_\pi(s,a) + (1-\varepsilon) \sum_{a \in A} \frac{\pi(a|s) - \frac{\varepsilon}{m}}{1-\varepsilon} q_\pi(s,a)$$

$$= \sum_{a \in A} \pi(a|s)\, q_\pi(s,a) = U_\pi(s)$$

# Monte-Carlo Policy Iteration

- sample episode(s) from the environment
- do MC policy evaluation to update q-values q(s,a)
- update $\pi'(s)$ based on the observed episodes and so on

remarks:

- it might not be necessary to update all q(s,a)
- using ε-greedy policy π′ makes sure that we explore underexplored action sufficiently at some point in time
- optimizing π′ is not the same as the optimal policy π*

=> How can we still find π*?

# GLIE

***Definition***:

Greedy in the Limit with Infinite Exploaration (GLIE)

- All state-action pairs are explored infinitely many times

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges to a greedy policy,

$$\lim_{k \to \infty} \pi_k(a|s) = 1\left(a = \operatorname*{argmax}_{a\prime \in A} Q_k(s, a')\right)$$

# GLIE Monte-Carlo Control

- Sample $k^{th}$ episode using π:

  $$x = S_1, R_1, A_1, \ldots, A_{t-1}, S_t, R_t \sim \pi$$

- For each state $S_t$ and action $A_t$ in x update:

  $$N_t(S_t, A_t) \leftarrow N_t(S_t, A_t) + 1$$

  $$Q_t(S_t, A_t) \leftarrow Q_t(S_t, A_t) + \frac{1}{N_t(S_t, A_t)}\left(G_t - Q(S_t, A_t)\right)$$

- Improve policy based on new action-value faction Q

  $$\epsilon \leftarrow \frac{1}{k}$$

  $$\pi \leftarrow \epsilon - greedy(Q)$$

**Theorem**: GLIE Monte-Carlo control converges to the optimal action-value function Q(s,a)→q*(s,a)

# TD and Monte-Carlo Control

In general TD has several advantages over MC:

- lower variance
- online learning (no waiting until end of episode)
- learning from incomplete episodes

How to control based on TD?

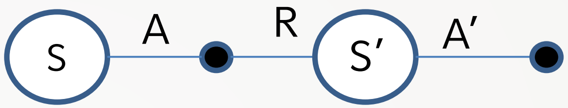
- Update Q(S,A) based on TD
- Use $\varepsilon$-greedy policy improvement
- Update every time step

# SARSA

TD applied to q-function Q(S,A):

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

requires a sample of form:




Basic idea behind SARSA (State Action Reward State Action):

For every step we do:

evaluate policy based on the above formula (policy evaluation)

update policy based on ε-greedy (policy evaluation)

# SARSA Algorithm

```
init Q(s,a)∀s ∈ S,a ∈ A and Q(terminal,)=0
for n episodes:
  init s
  choose a from A(s) based on Q(e.g.ε-greedy)
  repeat until episode is finished:
    s’,r = query_Env(s,a)
    choose a’from A(s’)based on Q(e.g.ε-greedy)
    Q(s,a)← Q(s,a)+α(r+γQ(S’,a’)-Q(s,a))
    s ← s’, a ← a’
  until s is terminal
```

# SARSA Convergence

**Theorem:** SARSA converges to the optimal action-value function q*, Q(S,A)->q*(s,a), if the following conditions hold:

- GLIE sequence of policies $\pi_t(a|s)$
- Robbins-Monro sequence of step-sizes $\alpha_t$

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$
$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

# n-step SARSA

- consider the following n-step q-values for n = 1,2,..,∞:

n=1 : $q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

n=2 : $q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})$

$\dots$

n=∞ : $q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$

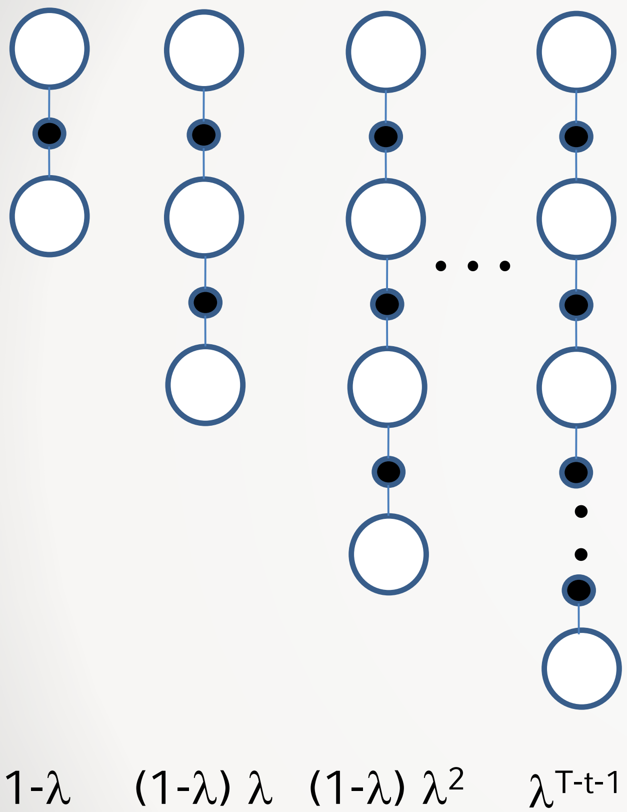
- general n-step Q-return:

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

- n-step temporal difference learning:

$$\mathrm{Q}(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left( q_t^{(n)} - \mathrm{Q}(S_t, A_t) \right)$$

# Forward View SARAS(λ)




- The $q^\lambda$-return combines all n-step Q-returns $q_t^n$
- Using weight $(1-\lambda)\,\lambda^{n-1}$ :

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, q_t^n$$

- Forward-view SARSA(λ):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(q_t^\lambda - Q(S_t, A_t)\right)$$

# Backward View SARSA(λ)

- Just as for TD(λ), we can employ eligibility traces
- However, SARSA(λ) needs an eligibility trace for each state-action pair instead for each state:

$$E_o(s,a) = 0$$

$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + 1(\mathrm{S}_t = s, \mathrm{A}_t = \mathrm{a})$$

- Q(s,a) is updated for every state s and action a
- Computing the error $\delta_t$ and eligibility trace $E_t(s,a)$ we can adapt the backward updates:

TD-error: $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$

TD-update: $Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$

# n-step SARSA Algorithm

```
init Q(s,a)∀s ∈ S,a ∈ A(s)
for n episodes:
  E(s,a)=0, ∀s ∈ S,a ∈ A(s)
  init S
  choose A from A(S)based on Q(e.g.ε-greedy)
  repeat until episode is finished:
    Take action A, observe R,S'
    choose A'from A(S')based on Q(e.g.ε-greedy)
    δ ← R + γQ(S',A') − Q(S,A)
    E(S,A) ← E(S,A)+1
    For ∀s ∈ S,a ∈ A(s):
        Q(s,a) ← Q(s,a) + αδ_t E_t(s,a)
         E_t(s,a) ← γλE_{t−1}(s,a)
    S←S',A ←A'
  until S is terminal
```

# Off-Policy Learning

**On-policy Learning**: Learn a policy by sampling from the same policy.

**But**: Sometimes it is better to observe behaviour of another policy to find a better policy.

$\Rightarrow$ **Off-Policy Learning**

- learn based on the experience of other agents (or humans)
- Re-use experience generated from old policies
- Learn the optimal policy by following an exploratory policy
- Learn about multiple policies while following one policy

# Importance Sampling

Can be compute the expectation of our returns when following another policy?

$\Rightarrow$ Rewards stay the same, but distribution over the states changes

$\Rightarrow$ We have to correct the likelihoods to adjust for the different visiting probabilities.

Importance sampling:

$$\mathbb{E}_{X\sim P}[f(X)] = \sum P(X)f(X) =$$

$$\sum Q(X)\frac{P(X)}{Q(X)}f(X) = \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]$$

# Importance Sampling for Off-Policy MC

- Consider the observed policy $\mu$ to evaluate target policy $\pi$
- Weight return $G_t$w.r.t. similarity between these policies
- Multiply importance sampling corrections along the whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1})}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1})} \dots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value with corrected return:

$$U(S_t) \leftarrow U(S_t) + \alpha\left(G_t^{\pi/\mu} - U(S_t)\right)$$

- Does not work if $\mu$=0 and $\pi$ >0
- Can significantly increase variance

# Importance Sampling for Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target R+γU(S') by importance sampling
- For TD only a single step correction is needed:

$$U(S_t) \leftarrow U(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\left(R_{t+1} + \gamma U(S_{t+1})\right) - U(S_t)\right)$$

- Much lower variance than MC importance sampling
- Policies need to be similar over a single step

# Q-Learning

- Off-policy learning of action-value pairs Q(s,a)
- No importance sampling is necessary
- Next action selected based on behaviour policy

$$A_{t+1} \sim \mu(\cdot, S_t)$$

- But updates are done on alternative successor:

$$A' \sim \pi(\cdot, S_t)$$

- And update $Q(S_t,A_t)$ towards value of alternate action:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right)$$

# Q-Learning

- Both policies the behaviour $\mu$ and the target $\pi$ policy are improved

- The target policy $\pi$ is greedy w.r.t. Q(s,a):

$$\pi(S_{t+1}) = \underset{a' \in A}{\operatorname{argmax}} Q(S_{t+1}, a')$$

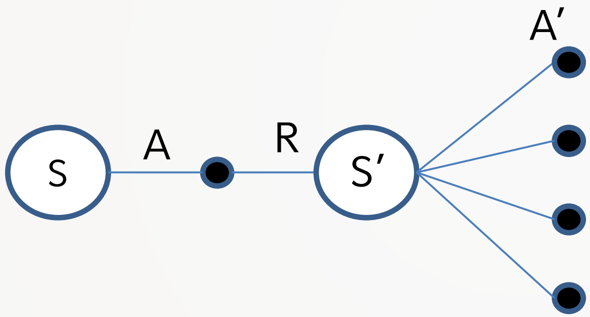
- The behaviour policy $\mu$ is e.g. $\varepsilon$-greedy w.r.t. Q(s,a)

- The Q-learning target the simplifies:

$$
\begin{aligned}
&R_{t+1} + \gamma Q(S_{t+1}, A') \\
&= R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a' \in A}{\operatorname{argmax}} Q(S_{t+1}, a')\right) \\
&= R_{t+1} + \max_{a' \in A} \gamma Q(S_{t+1}, a')
\end{aligned}
$$

# Q-Learning

**Theorem**: Q-learning control converges to the optimal action-value function, Q(S,A)→q*(s,a).




$$Q(S,A) \leftarrow Q(S,A) + \alpha \left( R + \gamma \max_{a' \in A} Q(S',a') - Q(S,A) \right)$$

# Q-Learning Algorithm

```
init Q(s,a)∀s ∈ S,a ∈ A
for n episodes:
  init s
  repeat until episode is finished:
    choose a from A(s) with π_b
    s’,r = query_Env(s,a)
    Q(s,a)← Q(s,a)+α(r+γmax_a Q(S’,a)-Q(s,a))
    s ← s’
  until s is terminated
//terminal state or finite horizon is reached
```

# Summary

- Policy Evaluation:
  - Monte-Carlo (MC) Learning: evaluate on complete episodes: high variance, no bias, slow convergence
  - Temporal Difference (TD) Learning: evaluate single steps and approximate future via Bootstrapping: lower variance, faster convergence, adds bias
  - TD($\lambda$): Links between MC and TD
- Control:
  - on-policy learning: "learn on the job"
    - GLIE Monte-Carlo Control
    - SARSA and SARSA($\lambda$)
  - off-policy learning: learn by watching other policies
    - MC and TD with importance sampling
    - Q-learning

# Literature

- Lecture notes D. Silver: Introduction to Reinforcement Learning (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)

- S. Russel, P. Norvig: Artificial Intelligence: A modern Approach, Pearson, 3rd edition, 2016

- R. S. Sutton, A. G. Barto: **Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning),** The MIT Press; Auflage: 2., 2018