# CE 273
# Markov Decision Processes

Lecture 18

## Approximation in Value Space - Part I

# Previously on Markov Decision Processes

Before we proceed, it is worthwhile to group the methods used in ADP. However, it is very difficulty to construct a taxonomy since the differences between various methods are often not pronounced.

Further, a majority of algorithms are heuristics and mix and match ideas from different solution approaches.

The different *techniques that we will discuss in this class* can be loosely be classified as

- Lookahead methods (Lecture 17)
- Approximations in value space (Lectures 18-20)
- Approximations in policy space (Lecture 21)

# Previously on Markov Decision Processes

Approximation in value space is the most popular approach in ADP. The goal in these methods to find a good approximation to the value function associated with a policy or the optimal value function.

Most algorithms in this class operate in a policy iteration format. The idea is to start with some policy $\mu$ and approximate $J_\mu$ as $\tilde{J}_\mu$. This is equivalent to the policy evaluation step in PI.

A policy 'improvement' step follows in which a new policy $\mu'$ is constructed using

$$\mu'(i) \in \arg \min_{u \in U(i)} \left\{ g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u) \tilde{J}_\mu(j) \right\}$$

And this process is repeated. The policy 'improvement' step is also referred to as choosing a greedy policy.

# Previously on Markov Decision Processes

Several methods to approximate value functions of a given policy exist. Two most commonly used approaches that we will see in this course are

- **Simulation-based approximation**
  Imagine we have a simulator that mimics our system (e.g., an Arena model of queues). To find $\tilde{J}_\mu(i)$, one could start the system with state $i$ and calculate the discounted cost of a trajectory and repeat this to estimate the the true $J_\mu(i)$ using sample averages.
- **Parametric methods**
  Another option is to represent the value function in a parameterized form and shift our attention to calculating the parameters that gives us the best fit (like regression). These parameters would however be different for different policies.

# Previously on Markov Decision Processes

A very useful value function-like concept in the context of RL is called Q-factors. Recall that the VI step takes the form

$$J_{k+1}(i) = \min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u) \Big( g(i,u,j) + \alpha J_k(j) \Big) \right\} \forall\, i = 1, \ldots, n$$

We define new iterates $Q_{k+1}(i,u) \,\forall\, i = 1, \ldots, n, u \in U(i)$ such that

$$Q_{k+1}(i,u) = \sum_{j=1}^{n} p_{ij}(u) \Big( g(i,u,j) + \alpha J_k(j) \Big)$$

Thus, we can rewrite VI algorithm as

$$J_{k+1}(i) = \min_{u \in U(i)} Q_{k+1}(i,u) \,\forall\, i = 1, \ldots, n$$

Can the VI algorithm be re-written in terms of the $Q$ values only

$$Q_{k+1}(i,u) = \sum_{j=1}^{n} p_{ij}(u) \Big( g(i,u,j) + \alpha \min_{v \in U(j)} Q_k(j,v) \Big)$$

# Previously on Markov Decision Processes

Notice that $Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha J^*(j)\Big)$. Thus, one can interpret $Q^*(i,u)$ as the value of taking an action $u$ in state $i$ and behaving optimally thereafter.

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v)\Big)$$

Similarly, we can define $Q_\mu(i,u)$ as the cost of using control $u$ in state $i$ and thereafter following policy $\mu$.

The equations for finding $Q_\mu(i,u)$ parallel that of $J_\mu$ but without the minimization operator.

$$Q_\mu(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha J_\mu(j)\Big) \; \forall \, i = 1, \ldots, n, u \in U(i)$$

The earlier expression $J_{k+1}(i) = \min_{u \in U(i)} Q_{k+1}(i,u)$ now becomes

$$J_\mu(j) = Q_\mu(j, \mu(j))$$

# Previously on Markov Decision Processes

Lookahead methods are approximation techniques in which some optimization is performed for a small and limited number of iterations/time steps.

For example, suppose we have an approximation $\tilde{J}$ of the optimal value function $J^*$. Then a *one-step lookahead policy* $\mu'$ is defined as

$$\mu'(i) \in \arg\min_{u \in U(i)} \left\{ g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u) \tilde{J}(j) \right\}$$

We expect that the lookahead policy $\mu'$ to be a good approximation to the optimal policy $\mu^*$.

Rollout is essentially a one-step lookahead method where $\tilde{J}$ is replaced with $J_\mu$ where $\mu$ is some policy that is easy to construct (say a myopic policy). It is also called the *base policy*.

Another way to interpret rollout is that we take the optimal action for one step assuming that we will revert to the base policy thereafter.

# Lecture Outline

1. Simulation-based Approximation
2. Parametric Methods

# Lecture Outline

## Simulation-based Approximation

# Simulation-based Approximation

## Introduction

Recall that, given a policy $\mu$, the goal is to find an approximation of $\tilde{J}_\mu$ of $J_\mu$ that can be used in an approximate PI method.

An obvious choice of approximating the value functions is to start from a state $i$ and follow policy $\mu$ and average the sample costs to get an estimate.

As discussed earlier, if we have a simulator, one can start it from different initial points and estimate the expected discounted cost of using the policy. Further, this process can be fully parallelized.

When do we stop the simulation of each trajectory? Depends on the discount factor.

If we have a large state space, the idea of sampling can be combined with parametric methods that we will discuss later.

# Simulation-based Approximation

## Monte Carlo Methods

Such simulation-based approximations are also called Monte Carlo methods.

There are two closely related issues with the earlier idea:

- Starting from a particular state may not be under our control (especially in online settings).
- We may never visit some states under some policies.

We will modify the method by just looking at sample trajectories instead of imposing conditions on the initial state. (This partly addresses the first issue.)

The second one, is a problem of exploration and it can give poor estimates for states that are rarely visited. We will address this in the next lecture.

# Simulation-based Approximation

## Monte Carlo Methods

The Monte Carlo method simulates trajectories from some initial state (say the initial board position of a game) and updates the value functions for **all states** that are visited in **each** trajectory.

Since we no longer have a choice on the initial state, we do not know when to stop despite discounting.

Hence, these methods are applicable only to situations in which trajectories always terminate (e.g., Optimal stopping problems, Stochastic shortest paths, Go). Such trajectories are also called **episodes**.

# Simulation-based Approximation

## First-Visit Monte Carlo Method

Consider a policy $\mu$. Suppose we simulate $S$ trajectories and each trajectory is indexed by $s$. A trajectory $s$ can be written as

$$i_0, \mu(i_0), i_1, \mu(i_1), \ldots, i_t, \mu(i_t), \ldots, i_{t(s)}$$

where $i_{t(s)}$ represents the terminal state of trajectory $s$. Given this trajectory, we can compute the sample future cost at every time step $t = 0, \ldots, t(s)$ as follows

$$G_t(s) = g(i_t, \mu(i_t), i_{t+1}) + \alpha g(i_{t+1}, \mu(i_{t+1}), i_{t+2}) + \ldots + \alpha^{t(s)-1-t} g(i_{t(s)-1}, \mu(i_{t(s)-1}), i_{t(s)})$$

Suppose we want to find the approximate value of some state $i$. Then, we look at the first occurrence of $i$ in each trajectory and calculate the discounted cost from that point onward.

This process is repeated for all trajectories and the costs are averaged. This method is also called **First-Visit Monte Carlo Method**.

# Simulation-based Approximation

## First-Visit Monte Carlo Method

Consider an MDP with four states where state 4 is a terminating state.

Suppose we generate four sample trajectories of the system with a given policy. The states are shown in blue and costs are in black.

$$1, 10, 3, 12, 1, 14, 4$$
$$2, 13, 3, 17, 2, 13, 3, 12, 1, 10, 3, 10, 4$$
$$1, 15, 2, 14, 1, 10, 3, 17, 2, 11, 4$$
$$3, 18, 3, 18, 3, 12, 1, 14, 4$$

Using the First-Visit Monte Carlo Method, find the approximate value function of a policy which generates the above trajectories and costs. Assume a discount factor of 0.5.

The sample expected discounted costs of starting from state 1 are 19.5, 15, 26.625, and 14.

# Simulation-based Approximation

## Pseudocode

**First-Visit Monte Carlo Method**

numVisits$(i) \leftarrow 0 \, \forall \, i = 1, \ldots, n$
totalCost$(i) \leftarrow 0 \, \forall \, i = 1, \ldots, n$
$\tilde{J}_\mu \leftarrow \mathbf{0}$
**for** $s = 1, \ldots, S$ **do**
    **for** $t = 0, 1, \ldots, t(s)$ **do**
        **if** $i_t$ has been visited for the first time in $s$ **then**
            totalCost$(i_t) \leftarrow$ totalCost$(i_t) + G_t(s)$
            numVisits$(i_t) \leftarrow$ numVisits$(i_t) + 1$
        **end if**
    **end for**
**end for**
$\tilde{J}_\mu(i) \leftarrow$ totalCost$(i)$/numVisits$(i) \, \forall \, i = 1, \ldots, n$

# Simulation-based Approximation

## Monte Carlo Methods

Convergence of the First-Visit Monte Carlo method follows naturally from the law of large numbers. The estimates of $J_\mu$ are also unbiased but may have some variance which shrinks to zero as the number of samples increase.

The earlier method looked at the first occurrence of a state in each sample trajectory/episode and calculated the total cost from that point.

As seen in the example, the same state can appear multiple times within a single sample trajectory.

Another commonly used variant, Every-Visit Monte Carlo method, exploits this feature by considering every visit to a state and averaging the total $G_t(s)$ value.

# Simulation-based Approximation

## Monte Carlo Methods

The approximate value function in the earlier algorithm was constructed only after observing all the sample trajectories.

Instead, one can also update them incrementally one sample at a time by computing running averages. For example, what is the average of $19.5, 15, 26.625$, and 14?

An alternate way of solving this is

$$\frac{19.5 + 15}{2} = 17.25$$

$$17.25 + \frac{1}{3}(26.625 - 17.25) = 20.375$$

$$20.375 + \frac{1}{4}(14 - 20.375) = 18.78$$

Mathematically, if we want to find the average of $x_1, \ldots, x_n$, find iterates $\mu_k = \frac{1}{k}\sum_{k=1}^{k} x_k$ in terms of previous $\mu_{k-1}$ as

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

# Simulation-based Approximation

## Monte Carlo Methods

Thus, one can incrementally update the approximate value functions for each sample trajectory $s$.

For every $i_t$ in the trajectory $i_0, \mu(i_0), i_1, \mu(i_1), \ldots, i_{t(s)}$, update

$$\text{numVisits}(i_t) \leftarrow \text{numVisits}(i_t) + 1$$

$$\tilde{J}_\mu(i_t) \leftarrow \tilde{J}_\mu(i_t) + \frac{1}{\text{numVisits}(i_t)}\Big(G_t(s) - \tilde{J}_\mu(i_t)\Big)$$

This method can be generalized as

$$\tilde{J}_\mu(i_t) \leftarrow \tilde{J}_\mu(i_t) + \gamma\Big(G_t(s) - \tilde{J}_\mu(i_t)\Big)$$

Comparing this with gradient descent methods, $\gamma$ can be interpreted as a step size and $G_t(s)$ can be thought of a target. One can let $\gamma$ shrink to zero over time.

This is ideal for scenarios in which the system dynamics are not time-invariant.

# Simulation-based Approximation

## Temporal-Difference Learning

There are two major drawbacks with Monte Carlo methods:

- The sample trajectories have to terminate. But not all problems have this feature.
- The entire trajectory has to be simulated for each sample to calculate costs.

The Temporal-Difference (TD) method avoids these issues by updating the value functions immediately after every transition of the Markov chain.

Instead of using $G_t(s)$ in the MC method, it replaces it with the observed cost $g(i_t, \mu(i_t), i_{t+1})$ and the current approximation of the discounted value function for the future $\alpha \tilde{J}_\mu(i_{t+1})$.

# Simulation-based Approximation

## Temporal-Difference Learning

Mathematically, the MC update

$$\tilde{J}_\mu(i_t) \leftarrow \tilde{J}_\mu(i_t) + \gamma\Big(G_t(s) - \tilde{J}_\mu(i_t)\Big)$$

is transformed to

$$\tilde{J}_\mu(i_t) \leftarrow \tilde{J}_\mu(i_t) + \gamma\Big(g(i_t, \mu(i_t), i_{t+1}) + \alpha\tilde{J}_\mu(i_{t+1}) - \tilde{J}_\mu(i_t)\Big)$$

This method is also called the TD(0) algorithm, and

- $g(i_t, \mu(i_t), i_{t+1}) + \alpha\tilde{J}_\mu(i_{t+1})$ is called the TD target
- $g(i_t, \mu(i_t), i_{t+1}) + \alpha\tilde{J}_\mu(i_{t+1}) - \tilde{J}_\mu(i_t)$ is called the TD error

Can you imagine how DP, MC, TD updates look using a decision tree representation?

The TD(0) algorithm exploits the Markov property unlike MC updates and hence can be applied after each transition.

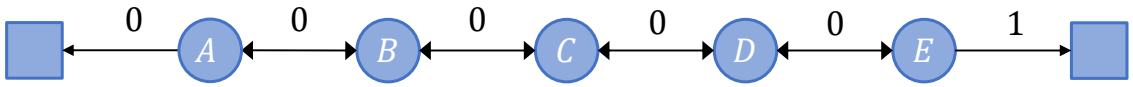
However, the estimates of the value function are biased since in each update we are using an approximation of the future value function.

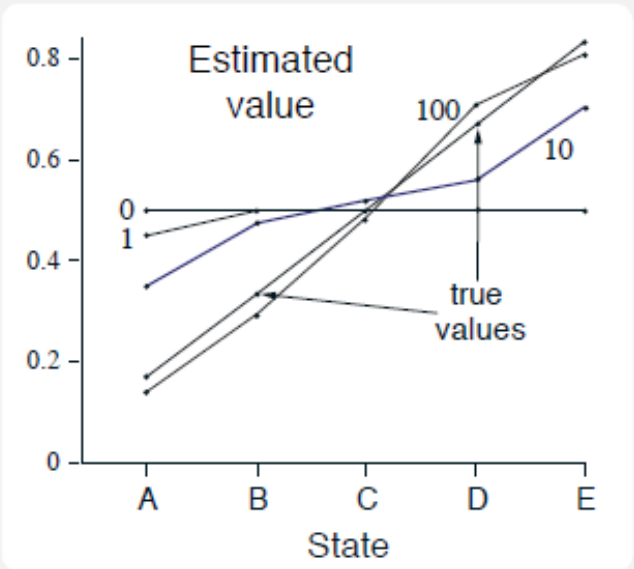
But the variance of the estimates are usually lower. (Why?) Empirically, TD methods are known to perform better than MC methods.

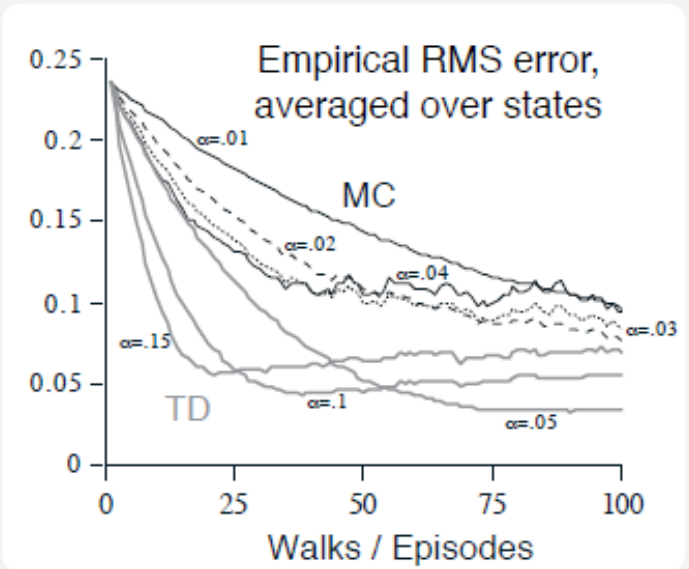
# Simulation-based Approximation

## Temporal-Difference Learning

Consider the following example from Sutton and Barto. There are five states $A, \ldots E$ and the square symbols represent terminating states. The one step transition costs are also indicated in the figure.



Both MC and TD methods have interesting interpretations which can be understood from the following example.

Suppose there are two states in an MDP, A and B, and we see the following episodes for some fixed policy:

- A, 0, B, 0
- B, 1
- B, 1
- B, 1

- B, 1
- B, 1
- B, 1
- B, 0

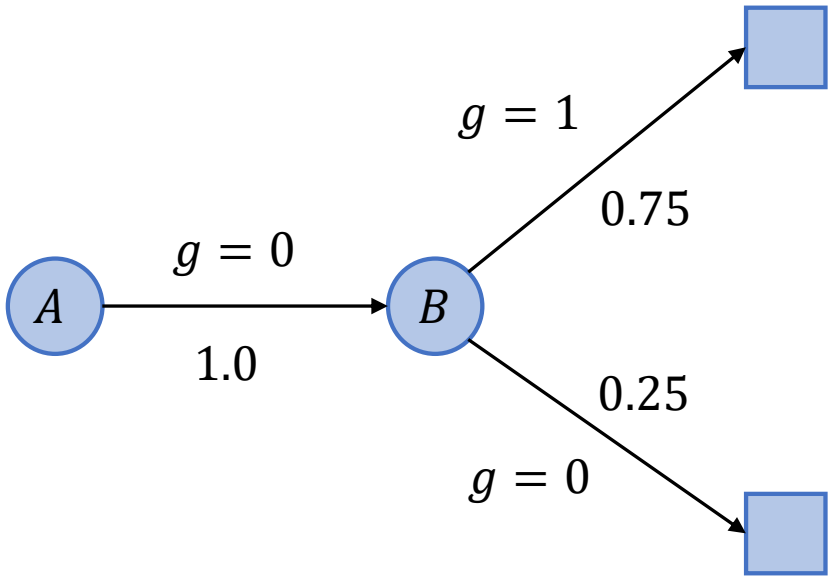
What are the approximate value functions at A and B?

# Simulation-based Approximation

## Temporal-Difference Learning

Thus, MC methods attempt to fit the value functions to the sample means and minimize the mean squared error.

TD methods on the other hand create a Markov chain by discovering transitions and costs along the lines of maximum-likelihood estimation and use it estimate approximate value functions.



# Simulation-based Approximation

## TD($\lambda$)

The target in the TD(0) algorithm was the 1-step cost and an estimate of the future costs $g(i_t, \mu(i_t), i_{t+1}) + \alpha \tilde{J}_\mu(i_{t+1})$.

Likewise, we can replace it with the 2-step costs and an estimate of the future costs after observing two transitions (but the calculations are computed at time $t$), i.e.,

$$G_t^{(2)}(s) = g(i_t, \mu(i_t), i_{t+1}) + \alpha g(i_{t+1}, \mu(i_{t+1}), i_{t+2}) + \alpha^2 \tilde{J}_\mu(i_{t+2})$$

Similarly, if we observe $n$ transitions, we can define a new target with $n$-step costs which looks like

$$G_t^{(n)}(s) = g(i_t, \mu(i_t), i_{t+1}) + \alpha g(i_{t+1}, \mu(i_{t+1}), i_{t+2}) + \ldots + \alpha^n \tilde{J}_\mu(i_{t+n})$$

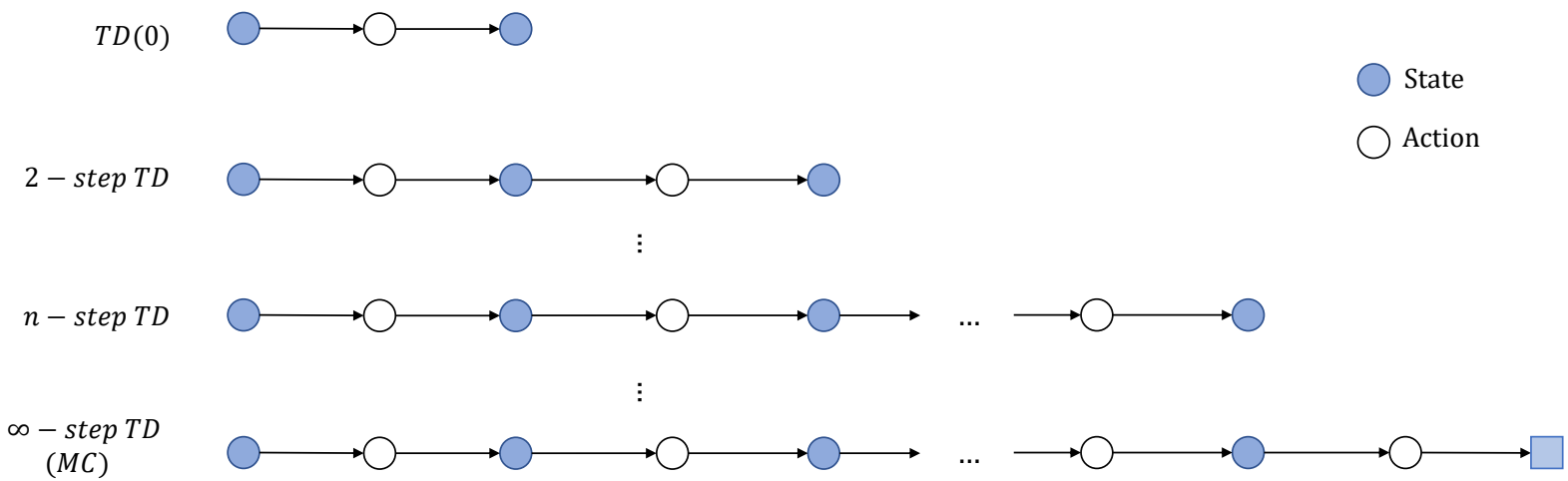
What happens when $n \to \infty$? $\infty$-step TD is equivalent to the MC method.

# Simulation-based Approximation

## TD($\lambda$)

The $n$-step TD algorithm can thus be written as

$$\tilde{J}_\mu(i_t) \leftarrow \tilde{J}_\mu(i_t) + \gamma\left(G_t^{(n)}(s) - \tilde{J}_\mu(i_t)\right)$$

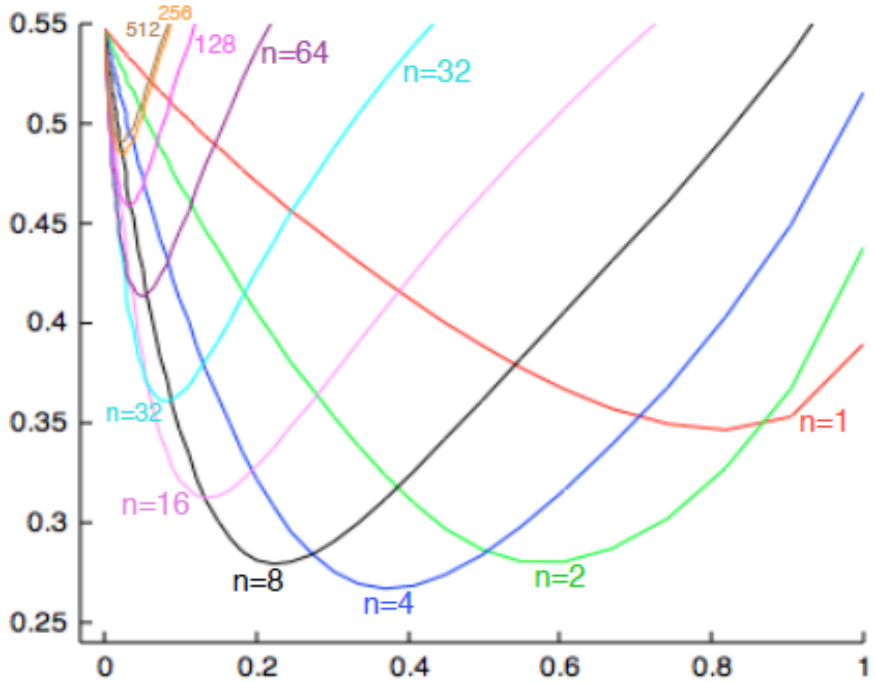



What is a good value of $n$?

# Simulation-based Approximation

## TD($\lambda$)

The following plot shows the performance of the $n$-step TD method on a 19 state random walk similar to the earlier example.

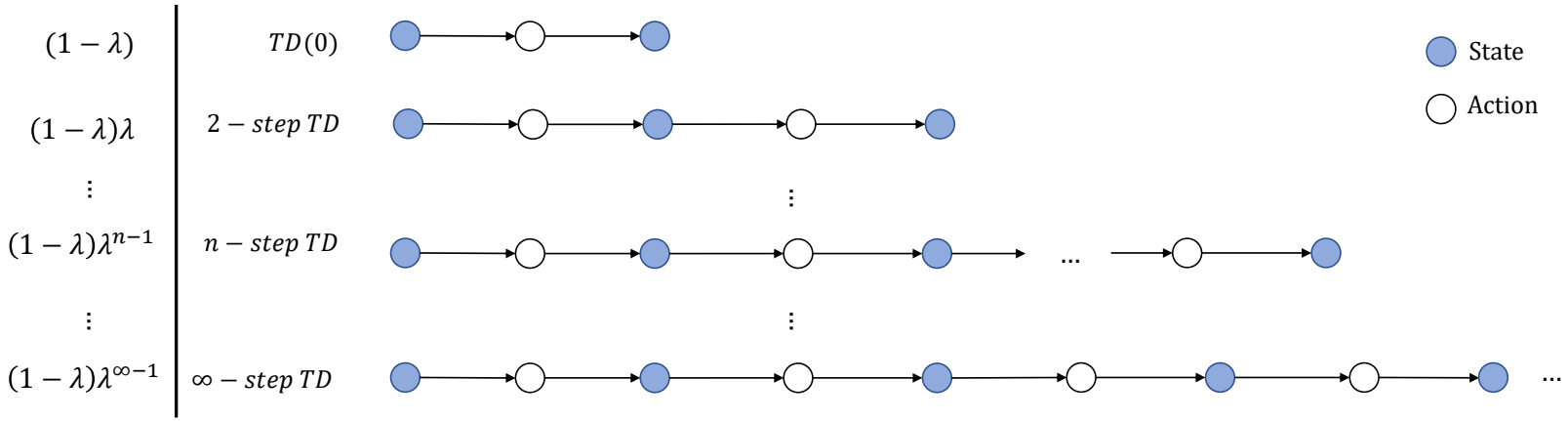



The $x$-axis represents $\gamma$ and the $y$-axis represents the average RMS error over all states for the first 10 episodes.

# Simulation-based Approximation

## TD($\lambda$)

In the TD($\lambda$) method, we modify the TD target by a weighted sum of all the $n$-step costs, using an parameter $\lambda$ which decays exponentially.




Mathematically,

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

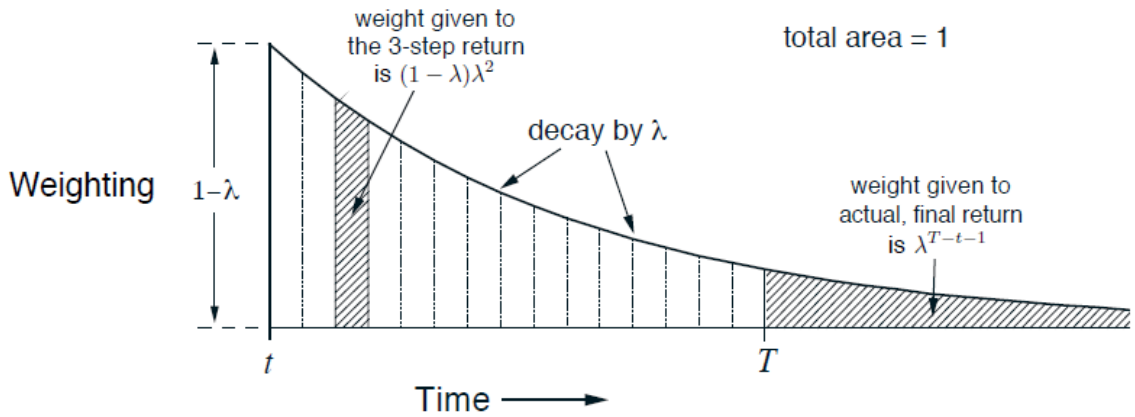
This weighted average reduces errors in the estimates of the value functions.

The algorithm TD($\lambda$) gets its name from the parameter $\lambda$. It is common to find algorithms in RL named after the notation used (e.g., Q-learning, SARSA).

# Simulation-based Approximation

## TD($\lambda$)

If the sample trajectory terminates, then the weights look as follows:




To compute the $n$-step costs, one has to wait for $n$ transitions. There are clever ways of avoiding this using what are called eligibility traces.

A celebrated application of the TD learning methods, particularly TD($\lambda$) can be found in

- Tesauro, G. (1995). Temporal difference learning and TD-Gammon. Communications of the ACM, 38(3), 58-68.

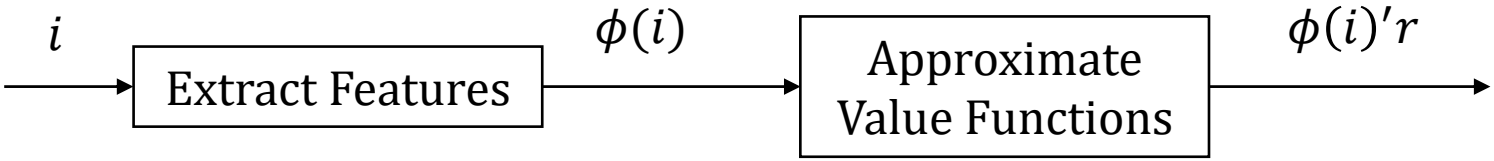
# Parametric Methods

# Parametric Methods

## Introduction

Recall that in parametric methods states are represented using features from an approximation architecture.

One then determines the relative weights of the features using a parameter vector $r$ such that $\tilde{J}_\mu(i; r)$ is close to $J_\mu(i)$. The features and weights are usually related in a linear way.




Suppose for each state $i$, we can find a column vector of features $\phi(i)$. The linear approximation architecture can be written as

$$\tilde{J}_\mu(i; r) = \phi(i)'r, \forall\, i = 1, \ldots, n$$

Suppose for each state $i$, we extract $m$ features. Let $k$ represent a generic feature.

Then, the vector of approximate value functions can be written as

$$\tilde{J} = \Phi r$$

where $\Phi$ is

$$\Phi = \begin{bmatrix} \phi_1(1) & \dots & \phi_m(1) \\ \phi_1(2) & \dots & \phi_m(2) \\ \vdots & \vdots & \vdots \\ \phi_1(n) & \dots & \phi_m(n) \end{bmatrix}_{n \times m} \qquad r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}_{m \times 1}$$

The rows of the $\Phi$ matrix are features and the columns can be interpreted as basis functions/vectors.

Thus, we can think of the subspace $S = \{\Phi r | r \in \mathbb{R}^m\}$ as the subspace spanned by the basis vectors (columns of $\Phi$).

# Parametric Methods

## Types

There are two main methods of finding the 'best' parameter vector $r$:

- **Direct**
  This is a simple function fitting exercise. If we know the true $J_\mu$, we can minmize the error between the function and its approximation.
- **Indirect**
  In this method we approximate the Bellman equation with what is called the projected Bellman equation. One way to think about it is that we are 'fitting' equations instead of functions.

# Parametric Methods

## Geometric Insight

Suppose we have access to the true value function $J_\mu$. Then, the optimal parameters can be obtained by solving

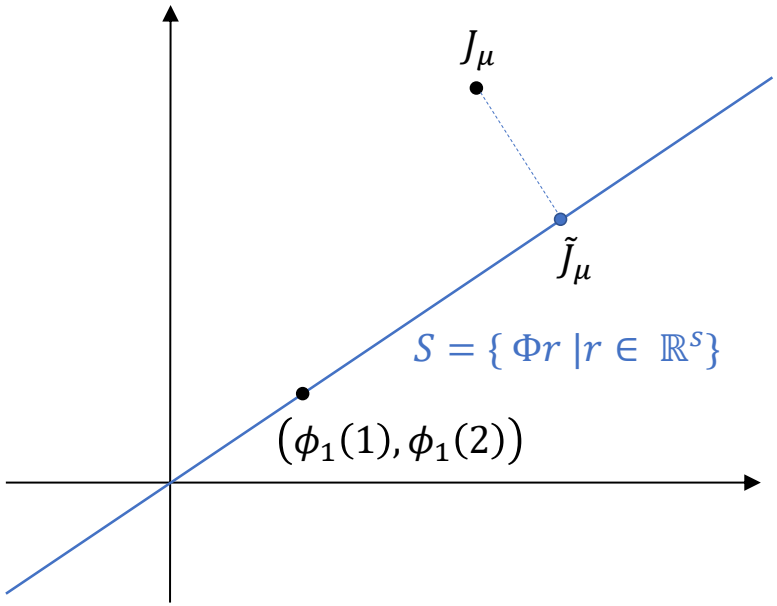
$$\min_{r \in \mathbb{R}^m} \|J_\mu - \Phi r\|$$

For illustration, it is easier to think of the value functions as vectors rather than functions. Suppose $J_\mu$ is a $2 \times 1$ vector and each state can be represented by 1 parameters.

How does the equation $\tilde{J}_\mu = \Phi r$ look and the above minimization look?

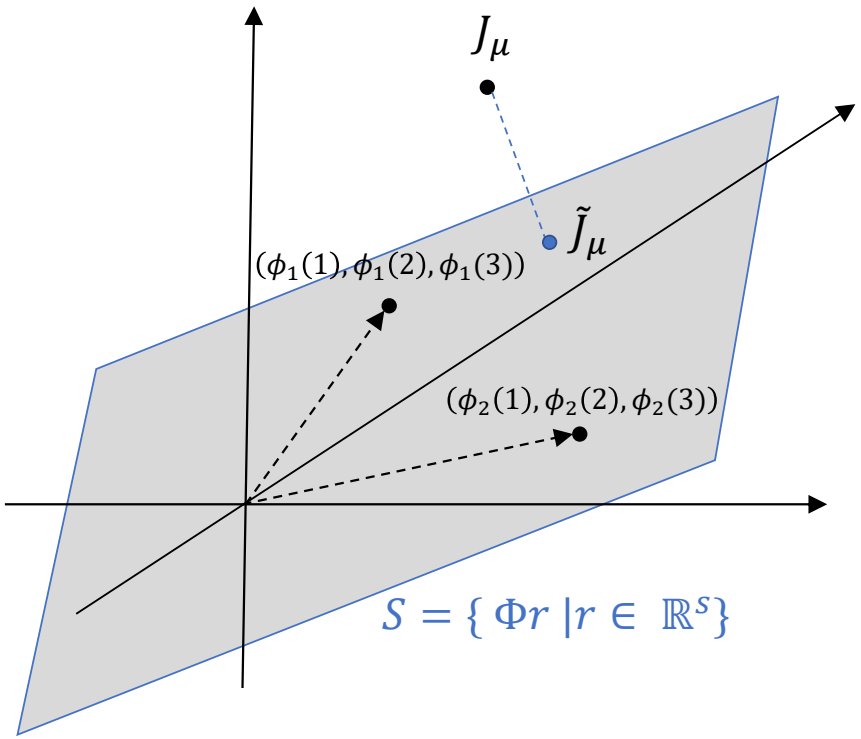
$$\tilde{J}_\mu = \begin{bmatrix} \tilde{J}_\mu(1) \\ \tilde{J}_\mu(2) \end{bmatrix} = \begin{bmatrix} \phi_1(1) \\ \phi_1(2) \end{bmatrix} r_1$$

Is $r$ uniquely determined?



Redraw this figure if $J_\mu$ and $r$ have dimensions $3 \times 1$ and $2 \times 1$ respectively.

$$\tilde{J}_\mu = \begin{bmatrix} \tilde{J}_\mu(1) \\ \tilde{J}_\mu(2) \\ \tilde{J}_\mu(3) \end{bmatrix} = \begin{bmatrix} \phi_1(1) & \phi_2(1) \\ \phi_1(2) & \phi_2(2) \\ \phi_1(3) & \phi_2(3) \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$$




Is $r$ uniquely determined?

# Parametric Methods

## Geometric Insight

In practice, we do not know $J_\mu$ and hence we will have to combine this method with a simulation-based approach.

This idea is closely related to the MC method we discussed earlier.

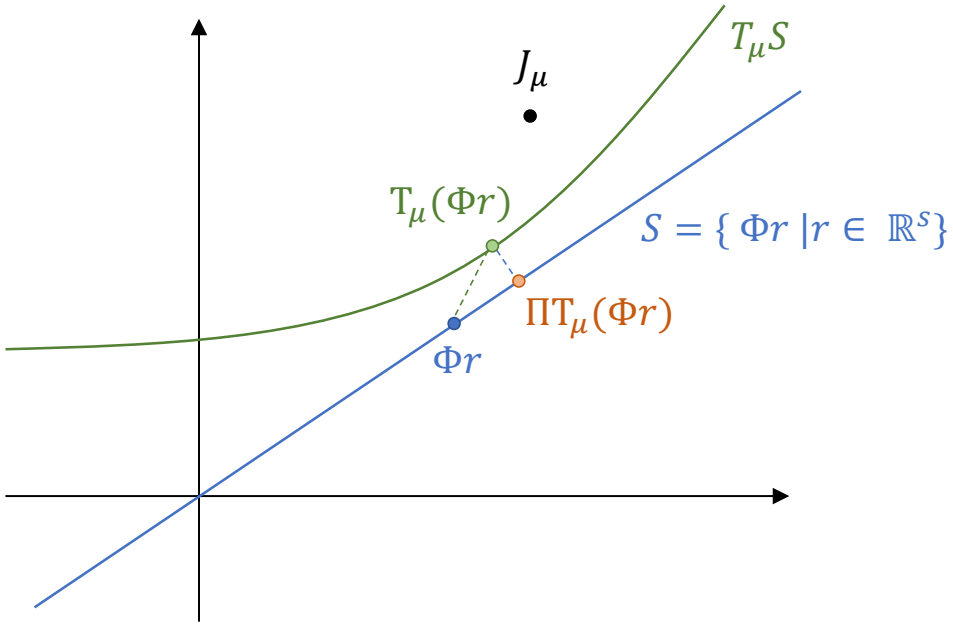
# Parametric Methods

## Geometric Insight

In the indirect approximation approach, we replace the Bellman equations $J_\mu = T_\mu J_\mu$ with

$$\Phi r = \Pi T_\mu(\Phi r)$$

where $\Pi$ denotes the projection of a point on the subspace $S$.




This approach has connections with the TD(0) method which we will discuss in detail in the next class.

# Your Moment of Zen