Brief Report

Approximate Dynamic Programming And The Krylov Subspace Methods

Ella Zhang *

Posted Date: 29 June 2023

doi: 10.20944/preprints202306.2088.v1

Keywords: Approximate Dynamic Programming; Krylov subspace method

Brief Report

Approximate Dynamic Programming and the Krylov Subspace Methods

Ella Zhang

Shandong Agricultural University

Abstract: This report discusses the application of approximate dynamic programming (ADP) and Krylov subspace methods in solving sequential decision-making problems in machine learning. ADP is used when dealing with large state spaces or when the exact dynamics are unknown. The paper explores various ADP methods such as temporal difference learning and least-squares policy evaluation. The authors also focus on the use of Krylov subspace methods for solving the Bellman equation in ADP. The report provides insights into linear approximation, stochastic algorithms, and the Arnoldi algorithm for orthonormal basis. Overall, the paper highlights the theoretical foundation provided by ADP and its connection with Krylov subspace methods, shedding light on their application in computer science.

Keywords: Approximate Dynamic Programming; Krylov Subspace Methods

1. Introduction

Dynamic programming (DP) is both an optimization method and a computer programming method which was developed by Richard Bellman in the 1950s. The main idea is to break a complicated problem down into simpler sub-problems recursively. If there exists a recursive relation between the original problem and its sub-problems, then DP methods are applicable. In the optimization literature this relationship is called the Bellman equation.

In machine learning, one of the problems that we are interested in is the sequential decision making problem regarding the way in which the software agents ought to take actions to maximize the cumulative rewards from the stochastic environment. We formulate the interaction bewteen agents and the environment by Markov Decision Process (MDP) whose structure can be modeled by the Bellman equation. But when the state space of the environment is too large or an exact model of the dynamics is absent, MDP can only be solved approximately with simulation and function approximation and this is what we call approximate dynamic programming (neuro-dynamic programming or reinforcement learning). A large body of methods have been developed, e.g. temporal difference learning (TD) [4] , least-squares temporal difference learning (LSTD) [3,4], least-squares policy evaluation (LSPE) [9] and least-squares policy iteration (LSPI) [6].

Computing the value function of a given policy (policy evaluation) by the Bellman equation plays an important role in solving approximate DP and it is equivalent to solving a linear system [13]. In matrix computations, Krylov subspace methods [5,8] are the most powerful tools to solve large scale linear systems of equations and least-squares problems. So there is a connection between these two areas but this connection lacks deep exploration.

Instead of using reinforcement learning, we use approximate DP in this report to emphasize its strong relationship with DP, because DP provides theoretical foundations and important guidelines for designing approximate DP algorithms.

My report will review some fundamental concepts and ideas of approximate DP and Krylov subspace methods and their connections.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

2. Approximate dynamic programming methods

2.1. Problem description

Markov decision process (MDP) is a framework to model the learning process that the agent learns from the interaction with the environment. Given the current state $S_t \in \mathcal{S}$ at time t, the agent interacts with the environment by selecting actions $A_t \in \mathcal{A}(S_t)$ according to the policy $\pi(A_t|S_t)$ and the environment would take the agent to another state $S_{t+1} \in \mathcal{S}$, and then provide a numerical feedback (immediate reward) $R_{t+1} = R(S_t, A_t, S_{t+1}) \in \mathbb{R}$ according to the dynamics (transition probability) $p(S_{t+1}, R_{t+1}|S_t, A_t)$. The policy π is a probability distribution from which we select the action. Our goal is to find an optimal policy (often deterministic) to maximize the long-term cumulative rewards. We assume that the policy is stationary. We only consider the inifite horizon discounted problem because the ideas can also be applied to the simpler absorbing Markov chains [17]. So the objective function is $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. It's easy to see the recursive structure

$$G_t = R_{t+1} + \gamma G_{t+1}. \tag{1}$$

The value function of the given policy π at state s is defined as

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\big],\ \forall s \in \mathcal{S} \tag{2}$$

It can be written as

$$\begin{aligned} v_\pi(s) &= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(A_t = a|S_t = s) \sum_{s'} \sum_r p(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)\big[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']\big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')],\ \forall s \in \mathcal{S} \end{aligned} \tag{3}$$

The equation (3) is the Bellman equation for MDP, which depicts the recursive relationships between the value of s and the value of its possible successor states s' and is consistent to (1). It can be written in a more compact form.

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \\ &= \sum_a \pi(a|s) \sum_{s',r} r \cdot p(s', r|s, a) + \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\gamma v_\pi(s') \\ &= \sum_a \pi(a|s) r(s, a) + \gamma \sum_{s'} \{\sum_a \pi(a|s) \cdot p(s'|s, a)\} v_\pi(s') \\ &= r_\pi(s) + \gamma \sum_{s'} p_\pi(s'|s) v_\pi(s'),\ \forall s \in \mathcal{S} \end{aligned} \tag{4}$$

The equation (4) can be written in the matrix form

$$v_\pi = r_\pi + \gamma P_\pi v_\pi \tag{5}$$

where $v_\pi \in \mathbb{R}^{|\mathcal{S}|}, r_\pi \in \mathbb{R}^{|\mathcal{S}|}$, $P_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ and $P_\pi(i, j) = p_\pi(j|i)$. Here $|\mathcal{S}|$ can be infinite, but for simplicity, we assume it is finite. Since we only consider a fixed policy, we can write (5) as $v = r + \gamma P v$ or $Lv = r$ where $L = I - \gamma P$.

We define a short hand notation for DP mapping:

$$Tv = r + \gamma P v,\ \forall v \in \mathcal{V} \tag{6}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

where $\mathcal{V}$ is the value function space and T is called the Bellman operator. It can be proved that T is a contraction mapping [1] and its fixed point v^* is the solution for (5). Finding the fixed point is exactly what we do in the policy evaluation step in the policy iteration method [13].

Solving (5) by a direct linear solver is not feasible when the state space is very large or when the transition matrix is unavailable. But we can find approximate solutions by value function approximation (VFA). Linear approximation is a popular VFA method [11] and it has the form of $\hat{v} = \Phi\theta, \Phi = (\Phi_1, \Phi_2, \ldots, \Phi_K) \in \mathbb{R}^{|\mathcal{S}|\times K}$, where Φ_j is the j-th feature function or basis function over the state space and K is the number of features, see [14]. For each state i, $\hat{v}(i) = \sum_{j=1}^{K} \Phi_j(i)\theta_j = \phi(i)^T\theta$, where $\phi(i)^T$ is the i^{th} row of Φ and represent the feature vector for state i. Linear function approximations have better convergence property [17] than nonlinear function approximation and is easier to analyze, interpret and implement. We can introduce nonlinearity when doing feature extraction, e.g. deep neural networks. Suppose we already have well-chosen features, then by the linear approximation, (5) becomes

$$\Phi\theta = r + \gamma P\Phi\theta. \tag{7}$$

We define a $\xi-$weighted quadratic norm on $\mathcal{V}$,

$$\|v\|_\xi = \sqrt{\sum_{s\in\mathcal{S}} \xi(s)(v(s))^2} = \sqrt{v^T\Xi v}, \ \forall v \in \mathcal{V} \tag{8}$$

where $\xi = (\xi_i)_{i=1}^{|\mathcal{S}|}$ with $\xi_i > 0$ is the weight vector and Ξ is the diagonal matrix with ξ on the diagonal. Then we can define the operator Π that projects any value function to its closest representable function in range(Φ) (i.e. column space of Φ) with respect to Ξ. That means Π is an orthogonal projector onto range(Φ) with respect to Ξ, i.e. $\Phi^T\Xi(v - \Pi v) = 0$, which is equivalent to

$$\Pi v \doteq \Phi\theta', \text{ where } \theta' = \arg\min_{\theta\in\mathbb{R}^K} \|v - \Phi\theta\|_\xi \tag{9}$$

We assume that the Markov chain we are studying is ergodic, then we have the stationary distribution,

$$\mu(s) = \lim_{N\leftarrow\infty} \frac{1}{N} \sum_{k=1}^{N} P(S_k = s|S_0 = s_0) > 0, \ \forall s, s_0 \in \mathcal{S} \tag{10}$$

We usually use the vector μ as our weight vector ξ, otherwise it may cause a *norm mismatch* problem which will make the iterative methods described in the following sections diverge [1,14].

Bertsekas [1] gives a short summary of the main approximate DP algorithms:

- Direct policy evaluation: Find θ that minimize $\|v - \Phi\theta\|_\xi^2$. We compute the approximation of the projection by Monte Carlo simulation. We can introduce nonlinear architectures in $\hat{v}$, e.g. deep neural networks.
- Indirect policy evaluation:
  - Solve the projected equation (Galerkin approximation [16,18])

    $$\Phi\theta = \Pi T(\Phi\theta) \tag{11}$$

    by simulation. Equation (11) is equivalent to minimizing the expected temporal difference error (TDE) $\|\hat{v} - \Pi T\hat{v}\|_\xi$ down to 0 for $\hat{v} \in$ range(Φ) [14].
    - TD learning: Stochastic iterative algorithm for solving (11).
    - Least-squares temporal difference learning (LSTD): TD fixed point method [14].
    - Least-squares policy evaluation (LSPE): A simulation-based form of projected iteration; essentially $\Phi\theta_{k+1} = \Pi T(\Phi\theta_k)+$simulation noise.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

– Bellman equation error (Bellman residual [14]) method: minimize $\|\Phi\theta - T(\Phi\theta)\|_\xi$

2.2. Direct policy evaluation

We are interested in finding the $\theta^* = \arg\min_{\theta\in\mathbb{R}^K}\|v - \hat{v}\|_\xi$ with $\hat{v} = \Phi\theta$. Suppose we can sample the exact values or give an accurate estimation of v, then we can learn the parameters in supervised learning paradigm. The samples or estimation of v could be considered as the true label, and $\hat{v}$ is the approximator. Then, it becomes a standard weighted least-squares problem. The solution is the projection of v onto range(Φ) with respect to Ξ [1]. Then,

$$\theta^* = \arg\min_{\theta\in\mathbb{R}^K}\|v - \Phi\theta\|_\xi^2 = \arg\min_{\theta\in\mathbb{R}^K}\sum_{i=1}^{|\mathcal{S}|}\xi_i(v(i) - \phi(i)^T\theta)^2$$

Set the gradient of θ to 0, we have

$$\theta^* = (\sum_{i=1}^{|\mathcal{S}|}\xi_i\phi(i)\phi(i)^T)^{-1}\sum_{i=1}^{|\mathcal{S}|}\xi_i\phi(i)v(i)$$

Suppose we have samples $\{(s_1, v(s_1)), \ldots, (s_m, v(s_m))\}$ generated by the environment or a simulator of the environment according to ξ, where $s_j \in \{1, \ldots, |\mathcal{S}|\}$ for $j = 1 : m$, then we can get the simulation-based approximation:

$$\hat{\theta}^* = \arg\min_{\theta\in\mathbb{R}^K}\sum_{j=1}^{m}(v(s_j) - \phi(s_j)^T\theta)^2 = (\sum_{j=1}^{m}\phi(s_j)\phi(s_j)^T)^{-1}\sum_{j=1}^{m}\phi(s_j)v(s_j)$$

Note that $\phi(s_j)\phi(s_j)^T$ is a $K \times K$ matrix. Thus, we are using low-dimensional linear algebra (of order K, the number of basis functions) compared with (5).

We can obtain the unbiased estimate of the true value function v by Monte Carlo sampling, i.e. perform long-term lookahead through the Markov chain and collect all the future rewards. However, the Monte-Carlo estimate of v requires a large number of samples as it suffers from a high variance.

2.3. Indirect policy evaluation

2.3.1. Stochastic algorithms: a general view

Consider the approximate solution $\hat{x}_m$ of a linear system of equations $x = b + Ax,\ A \in \mathbb{R}^{n\times n}$ by using m simulation samples $b + w_k$ and $A + W_k,\ k = 1, \ldots, m$, where $w_k,\ W_k$ are simulation noise. Think of $x = b + Ax$ as approximate policy evaluation which is the same as (5) [1]:

- Stochastic approximation (SA) approach: starting from an initial point x_0, for $k = 1, \ldots, m$

$$x_k = (1 - \alpha_k)x_{k-1} + \alpha_k((b + w_k) + (A + W_k)x_{k-1}) \tag{12}$$

  $\alpha_k,\ k = 1, \ldots, m$ are the weights.
- Monte Carlo estimation (MCE) approach: Form Monte Carlo estimation of b and A,

$$b_m = \frac{1}{m}\sum_{k=1}^{m}(b + w_k), \quad A_m = \frac{1}{m}\sum_{k=1}^{m}(A + W_k) \tag{13}$$

  Then solve $x = b_m + A_m x$ by a direct method or an iterative method. $A_m,\ b_m$ can be built incrementally.

TD is SA method, LSTD and LSPE are MCE methods. We will introduce them in section 2.3.2-2.3.4.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

2.3.2. Temporal difference (TD) learning

In 1988, Sutton developed temporal difference learning as a class of incremental learning methods for general prediction problems. It assigns credit through the difference of the predictions between current state and its successor, which makes it different with the conventional supervised learning methods. It has the form [4]

$$v(S_t) = v(S_t) + \alpha[R_{t+1} + \gamma v(S_{t+1}) - v(S_t)] \tag{14}$$

where α is a constant step-size parameter. This form is equivalent to (12). TD learning is a combination of Monte Carlo ideas and DP ideas. It can learn directly from the experience R_{t+1} sampled from environment which is similar to Monte Carlo methods. It also updates the estimates partially based on other learned estimates $v(S_{t+1})$ without collecting all future rewards (bootstrapping) which is similar to DP.

When we use linear function approximation $\hat{v}(s_t) = \phi(s_t)^T\theta_t$, the parameter update can be written as [4],

$$\theta_{t+1} = \theta_t + \alpha[R_{t+1} + \gamma\phi(s_{t+1})^T\theta_t - \phi(s_t)^T\theta_t]\phi(s_t) \tag{15}$$

Equation (15) is like we are doing $\min_\theta J(\theta) = (v(s_t) - \hat{v}(s_t))^2$. We apply gradient descent $\theta_{t+1} = \theta_t - \frac{1}{2}\alpha\nabla J(\theta) = \theta_t + \alpha(v(s_t) - \hat{v}(s_t))\nabla\hat{v}(s_t)$ and substitute $v(s_t)$ by its approximation $R_{t+1} + \gamma\phi(s_{t+1})^T\theta_t$. We call them semi-gradient methods because we only take derivative of $\hat{v}(s_t)$ but not of $R_{t+1} + \gamma\phi(s_{t+1})^T\theta_t$. TD is proved to be convergent with probability one [17].

TD requires less memory, less peak computation and produce more accurate predictions than conventional supervised learning methods.

2.3.3. Least-squares temporal difference (LSTD) learning

As ΠT is a contraction mapping on range(Φ) with respect to Ξ [1,14], there exists an unique fixed point $\Phi\theta^*$ that satisfies (11). That means θ^* is the unique solution to

$$\Phi\theta = \Pi T(\Phi\theta) \tag{16}$$

which means $\Phi\theta^*$ is the unique point in range(Φ) that satisfies the condition that when we use the projector ΠT on it, the projection is itself. According to the definition of Π, θ^* should satisfies the condition

$$\Phi^T\Xi(\Phi\theta^* - (r + \gamma P\Phi\theta^*)) = 0. \tag{17}$$

The orthogonal condition (17) is equivalent to the Galerkin method [16,18]. Equation (17) can be written as a linear systems of equations $C\theta^* = d$, where

$$C = \Phi^T\Xi(I - \gamma P)\Phi, \quad d = \Phi^T\Xi r \tag{18}$$

So we can have $\theta^* = C^{-1}d$. The simulation machanism for C and d is

$$d_k = \frac{1}{k+1}\sum_{t=0}^{k}\phi(s_t)r_t \approx \sum_{i,j}\xi_i P(i,j)\phi(i)R(i,j) = \Phi^T\Xi r = d \tag{19}$$

$$C_k = \frac{1}{k+1}\sum_{t=0}^{k}\phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T \approx \Phi^T\Xi(I - \gamma P)\Phi = C \tag{20}$$

where $C_k \in \mathbb{R}^{K\times K}$, $d_k \in \mathbb{R}^K$ can be computed with low-dimensional calculation. Then θ^* can be approximated by $\hat{\theta}_k = C_k^{-1}d_k$.

LSTD makes more efficient use of data because it extract more information from each additional observation, and thus would be expected to converge with fewer training samples. In addition, it does

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

not require the user to manually tune a stepsize schedule for good performance. LSTD converges to the same point as TD [4].

TD's convergence can be slowed dramatically by a poor choice of the stepsize parameters α. Furthermore, TD's performance is sensitive to the initial estimate for θ. LSTD does not rely on an arbitrary initial estimate. TD is also sensitive to the ranges of the individual features. LSTD is not.

2.3.4. Least-squares policy evaluation

An alternative way to solve the projected equation is projected value iteration (PVI), that is

$$\Phi\theta_{k+1} = \Pi T(\Phi\theta_k) = \Pi(r + \gamma P\Phi\theta_k) \tag{21}$$

It converges to θ^* because ΠT is a contraction mapping with respect to Ξ. Rather than directly solve the projected equation, it uses an iterative way to get close to the fixed point starting from an arbitrary initial point. It is equivalent to the minimization problem

$$\theta_{k+1} = \arg\min_{\theta\in\mathbb{R}^K} \left\| \Phi\theta - (r + \gamma P\Phi\theta^k) \right\|_\xi^2 \tag{22}$$

Set the gradient with respect to θ to be 0, we could have

$$\Phi^T\Xi(\Phi\theta_{k+1} - (r + \gamma P\Phi\theta_k)) = 0 \tag{23}$$

Then we could have

$$\theta_{k+1} = \theta_k - (\Phi^T\Xi\Phi)^{-1}(C\theta_k - d) \tag{24}$$

where C, d are defined as (18) and could be estimated by (19) and (20). $G = \Phi^T\Xi\Phi$ could be estimated by

$$G_k = \frac{1}{k+1}\sum_{t=0}^{k} \phi(s_t)\phi(s_t)^T \approx \Phi^T\Xi\Phi = G \tag{25}$$

C_k, d_k and G_k can be formed incrementally or in temporal difference style [1].

2.3.5. Bellman equation error methods

The main idea of the Bellman equation error (BE) method is to find a $\hat{v} \in \text{range}(\Phi)$ so that $\|\hat{v} - T\hat{v}\|_\xi$ can be minimized. Recall that $L = I - \gamma P$. With the notation $\Psi = L\Phi$, we have $\|\hat{v} - T\hat{v}\|_\xi = \|\Phi\theta - r - \gamma P\Phi\theta\|_\xi = \|\Psi\theta - r\|_\xi$. Then it becomes a standard least squares problem with regard to $\|\cdot\|_\xi$. It is easy to see that the solution to the weighted least squares problem is

$$\theta_{BE} = (\Psi^T\Xi\Psi)^{-1}\Psi^T\Xi r \tag{26}$$

One reason that we are interested in BE method is the well-known result: $\forall\hat{v},\ \|v - \hat{v}\|_\infty \leq \frac{1}{1-\gamma}\|T\hat{v} - \hat{v}\|_\infty$ and for weighted norm we have $\forall\hat{v},\ \|v - \hat{v}\|_\xi \leq \frac{\sqrt{C(\xi)}}{1-\gamma}\|T\hat{v} - \hat{v}\|_\xi$, where $C(\xi) := max_{i,j}\frac{P(i,j)}{\xi_i}$ is a "concentration coefficient", which can be seen as a measure of the stochasticity of the MDP [14]. This means when BE is small, we are close to the true value function v. But for TD, there is no similar result. Actually, it is proved [14] that BE is an upper bound of the TDE. More exactly, since $\hat{v} = \Phi\theta = \Pi\Phi\theta$ and from Pythagorean Theorem,

$$BE^2 = \|\hat{v} - T\hat{v}\|_\xi^2 = \|\Pi(\hat{v} - T\hat{v})\|_\xi^2 + \|(I - \Pi)(\hat{v} - T\hat{v})\|_\xi^2 = TDE^2 + \|T\hat{v} - \Pi T\hat{v}\|_\xi^2 \tag{27}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

which means minimizing BE is not only minimizing TDE, but also minimizing $\|T\hat{v} - \Pi T\hat{v}\|_\xi^2$, which is part that the TD method misses. And also we can see that $TDE^2 = \|\Pi(\hat{v} - T\hat{v})\|_\xi^2$ and this is why people also call it the projected Bellman error (PBE).

BE has the additional component which has to do with the features. Hence, if the feature space is fixed, the term we really want to minimize is TDE. Only when features can be modified we start to look at BE.

2.4. Least-squares policy iteration (LSPI)

Motivated by LSTD, Lagoudakis [6] extended it to control problems. Instead of evaluating the state value function v, he did evaluation on the state-action pairs (s, a), i.e. the Q function. The resulting method is referred to as LSTDQ. The Q function is defined as

$$Q(s,a) = E\big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\big],\ \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{28}$$

We can directly choose action and improve our policy from the Q function without any knowledge of the underlying model of the environment. Everything about LSTDQ looks similar to LSTD after we change Φ to be the feature matrix for all state-action pairs. Then we put LSTD into the general policy iteration (GPI) frameworks to update our policy and get LSPI algorithm.

2.5. A unified oblique projection view

In the LSTD, we attempt to find the fixed point of the composite projector ΠT, where Π is an orthogonal projection onto range(Φ) with respect to Ξ. Now we consider using a projector Π that is not necessarily orthogonal, which is a general operator that satisfies $\Pi^2 = \Pi$ and whose range is range(Φ). In the most general case, such operators are called oblique projections and can be written $\Pi_X = \Phi\pi_X$ with $\pi_X = (X^T\Phi)^{-1}X^T$. The parameter X specifies the projection direction: precisely, Π_X is the projection onto range(Φ) orthogonal to range(X) or we could write it as $\forall v,\ (v - \Pi_X v) \perp X$ (the Π defined in (9) can be written as $\Pi_{\Xi\Phi}$). The vector $\pi_X v$ is the coordinate of the projection of v under the basis Φ.

Scherrer [14] provided a unified oblique projection view of TD fixed point and BE minimization methods. Recall that $\theta_{TD} = (\Phi^T\Xi L\Phi)^{-1}\Phi^T\Xi r,\ \theta_{BE} = (\Psi^T\Xi\Psi)^{-1}\Psi^T\Xi r$. If we write $X_{TD} = \Xi\Phi$ and $X_{BE} = \Xi L\Phi$, we can have (1) The TD fixed point and the BE minimizer are solutions to the projected equation $\hat{v}_X = \Pi_X T\hat{v}_X$ ($X = X_{TD}$ and $X = X_{BR}$ respectively). (2) When it exists, the solution of this projected equation is the projection of v onto range(Φ) orthogonally to range$(L^T X)$, i.e. formally $\hat{v}_X = \Pi_{L^T X} v$. From this perspective, we put the two methods under the same framework of projected fixed point equation.

From the results above, it's easy to have

$$\|v - \hat{v}_X\|_\xi \le \|\Pi_{L^T X}\|_\xi \|v - \Pi v\|_\xi = \sqrt{\sigma(ABCB^T)}\|v - \Pi v\|_\xi,\ \forall X. \tag{29}$$

where $A = \Phi^T\Xi\Phi,\ B = (X^T L\Phi)^{-1}, C = X^T L\Xi^{-1}L^T X$ are matrices of size $K \times K$ and σ is the spectral radius of the given matrix. A, B, C can be estimated by simulations. This inequality could also give us some inspiration on how to choose X and extract Φ, e.g. if we choose $X = (L^T)^{-1}\Xi\Phi$, we could have $\hat{v} = \Pi_{\Xi\Phi} v$, which is the orthogonal projection of true v onto range(Φ) with respect to Ξ .

Finally, Scherrer gives a comparison of TD and BE methods: BE is more robust than TD because it has guarantee of its performance. Although TD often outperforms BE, it could sometimes fail dramatically. So TD is more risky. In practice, TD is easier to implement by sampling and has lower variance. BE needs more samples to eliminate the effect of noise.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

3. Krylov subspace methods

The earliest Krylov subspace methods were developed by Lanczos, Hestenes and Stiefel in the late 1940s and early 1950s for solving large sparse linear systems of equations $Ax = b$ (with nonsingular $A \in \mathbb{R}^{n\times n}$). Krylov subspace methods form a the class of iterative method. An iterative method that solves the linear systems by producing a sequence of iterates $\{x^{(k)}\}$ starting from an initial point $x^{(0)}$, which is expected to converge to the true solution. Iterative methods have advantages over the direct methods (e.g. matrix factorization methods) when A is a large sparse matrix whose structure is hard to exploit. Moreover, we can stop an iterative method early if we just need an approximate solution, but we cannot do this with a direct method. Krylov subspace methods are different from classical iterative methods (e.g. matrix splitting methods) because they have memory, i.e. they construct $x^{(k+1)}$ not only from $x^{(k)}$ but also from the information of the previous iterates. So they are faster. Furthermore, Krylov subspace methods require only the ability to determine Ax for any x (some also require the computation of $A^T x$).

3.1. Projection Process

First we will introduce a general projection process to find the solution $\hat{x}$. Let x_0 denote an initial approximation to $\hat{x}$ and then the initial residual $r_0 \equiv b - Ax_0$. At step k, we impose some conditions on x_k:

$$x_k \in x_0 + \mathcal{S}_k,\ r_k \equiv b - Ax_k \perp \mathcal{C}_k,\ k = 1, 2, \cdots. \tag{30}$$

where $\mathcal{S}_k \subseteq \mathbb{R}^n$ is a k-dimensional subspace, referred to as the search subspace, and $\mathcal{C}_k \subseteq \mathbb{R}^n$ is also a $k-$dimensional subspace, referred to as the constraint subspace. Since S_k has dimension k, we have k degrees of freedom to construct x_k. In order to determine x_k, we need $\mathcal{C}_k$ to give us k constraints which are imposed on the residual r_k in (30).

Let $V_k \in \mathbb{R}^{n\times k}$ and $U_k \in \mathbb{R}^{n\times k}$ be two full-column rank matrices such that

$$\mathcal{S}_k = \text{range}(V_k), \quad \mathcal{C}_k = \text{range}(U_k) \tag{31}$$

Then from (1) there exists $z_k \in \mathbb{R}^k$ such that

$$x_k = x_0 + V_k z_k \tag{32}$$

and

$$U_k^T(b - Ax_0 - AV_k z_k) = 0 \tag{33}$$

which can be rewritten as

$$U_k^T AV_k z_k = U_k^T r_0 \tag{34}$$

Theorem 1. *1(see Theorem 2.1.2 in [8]) The matrix $U_k^T AV_k$ in equation (34) is nonsingular if and only if*

$$\mathbb{R}^n = A\mathcal{S}_k \oplus \mathcal{C}_k^{\perp} \tag{35}$$

Now we assume that (35) holds. The linear system (34) can be regarded as a projected one of the original linear system $Ax = b$. As the dimensions of the matrix $U_k^T AV_k$ is $k \times k$, (34) can also be regarded as a reduced linear system.

From (32) and (34)

$$x_k = x_0 + V_k(U_k^T AV_k)^{-1} U_k^T r_0 \tag{36}$$

Then

$$\begin{aligned} r_k &= b - Ax_k \\ &= b - Ax_0 - AV_k(U_k^T AV_k)^{-1} U_k^T r_0 = (I - P_k)r_0 \end{aligned}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

where

$$P_k \equiv AV_k(U_k^T AV_k)^{-1}U_k^T \tag{37}$$

The matrix P_k is an oblique projector because $P_k^2 = P_k$. So r_k is the projection of r_0 onto $\mathcal{C}_k^{\perp}$. And it is easy to verify that in (35) $A\mathcal{S}_k = \mathcal{R}(P_k)$ and $\mathcal{C}_k^{\perp} = \mathcal{N}(P_k) = \mathcal{R}(I - P_k)$.

Theorem 2. *2 (see Lemma 2.1.7 in [8]) Suppose that (34) holds. If $r_0 \in \mathcal{S}_k$ and $A\mathcal{S}_k = \mathcal{S}_k$, then x_k is the solution of $Ax = b$*

Theorem 2 gives a sufficent condition for x_k to be the solution of $Ax = b$ at step k, that is in order to to make $r_k = 0$ we need to have $r_0 = P_k r_0$ [8]. It seems like a good idea to start the projection process with the search space $\mathcal{S}_1 = \text{span}\{r_0\}$, and to build up a nested sequence of search spaces.

$$\text{span}(r_0) = \mathcal{S}_1 \subseteq \mathcal{S}_2 \subseteq \mathcal{S}_3 \subseteq \cdots \tag{38}$$

and these spaces eventually satisfy $A\mathcal{S}_k = \mathcal{S}_k$ for some k. Then we can obtain the solution in finite number of steps and we hope $k \ll n$.

3.2. Krylov Subspaces

Given $A \in \mathbb{R}^{n\times n}$ and a nonzero vector $v \in \mathbb{R}^n$, consider the following Krylov sequence,

$$v,\ Av,\ A^2v, \ldots. \tag{39}$$

There exists an integer $d = d(A, v)$ such that $\{v, \ldots A^{d-1}v\}$ are linearly independent, but $\{v, \ldots A^d v\}$ are linearly dependent. It's easy to see that $d < n$. The matrix $\mathcal{K}_k(A, v) = \text{span}[v, Av, \ldots A^{k-1}v]$ is called a Krylov subspace. We will show that $d \leq m$, where m is the degree of the minimal polynomial of A to be introduced below.

The characteristic polynomial of matrix A is defined by

$$P_A(t) = det(tI - A) = \prod_{i=1}^{k}(t - \lambda_i)^{s_i} \tag{40}$$

where $\lambda_1, \cdots, \lambda_k$ are distinct eigenvalues of A and s_i is the algebraic multiplicity for the corresponding λ_i. The minimal polynomial of A is defined as

$$m_A(t) = \prod_{i=1}^{k}(t - \lambda_i)^{m_i} \tag{41}$$

where m_i is the largest size of Jordan block associated with λ_i. We all know that for a Jordan block J_i with eigenvalue λ_i whose size is m_i, $(J - \lambda_i I)^{m_i} = 0$ and $(J - \lambda_i I)^{m_i - 1} \neq 0$. And for a polynomial $q(t)$ and a matrix B whose Jordan blocks are $B_1, \cdots, B_k$, $q(B)$ has Jordan blocks $q(B_1), \cdots, q(B_k)$. Then for matrix A whose minimal polynomial is $m_A(t)$, we have $m_A(A) = 0$. Then for all v, we have $m_A(A)v = 0$. $m_A(t)$ could be written as

$$m_A(t) = \prod_{i=1}^{k}(t - \lambda_i)^{m_i} = \sum_{j=0}^{m}\alpha_j t^j \tag{42}$$

where $m = \sum_{i=1}^{k} m_i$. It it easy to see that $\alpha_m = 1$ from (41), so $m_A(A)v = 0$ means $A^m v$ can be represented by $v, Av, \ldots, A^{m-1}v$ which means $A^m v \in \mathcal{K}_m(A, v)$ and we must have $d \leq m$. Combining this conclusion with theorem 2 and the fact that $r_0 \in \mathcal{K}_i(A, r_0)$, it tells us that $\mathcal{K}_i(A, r_0)$ will eventually

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

stop growing and become invariant under A and $\mathcal{K}_i(A, r_0),\ i = 1, \cdots, m$ is a choice of nested sequence of spaces satisfy (38). So we have

Theorem 3. *3 If the minimal polynomial of nonsingular matrix A has degree m, then the solution to $Ax = b$ lies in the space $\mathcal{K}_m(A, b)$.*

Actually, with different choice of $\mathcal{S}_k$ and $\mathcal{C}_k$, we have different Krylov subspace methods as Table 1 shows.

Table 1. Different Krylov subspace methods.

Name	Problem	$\mathcal{S}_k$	$\mathcal{C}_k$	Property
CG	SPD[1] A	$\mathcal{K}_{k(A,r_0)}$	$\mathcal{K}_{k(A,r_0)}$	$x_k = \arg\min_{x_0+\mathcal{K}_{k(A,r_0)}} \|x - x_*\|_A$
SYMMLQ	$A^T = A$	$A\mathcal{K}_{k(A,r_0)}$	$\mathcal{K}_{k(A,r_0)}$	$x_k = \arg\min_{x_0+\mathcal{K}_{k(A,r_0)}} \|x - x_*\|_2$ [2]
MINRES	$A^T = A$	$\mathcal{K}_{k(A,r_0)}$	$A\mathcal{K}_{k(A,r_0)}$	$x_k = \arg\min_{x_0+\mathcal{K}_{k(A,r_0)}} \|b - Ax\|_2$
GMRES	general A	$\mathcal{K}_{k(A,r_0)}$	$A\mathcal{K}_{k(A,r_0)}$	$x_k = \arg\min_{x_0+\mathcal{K}_{k(A,r_0)}} \|b - Ax\|_2$
BiCG	general A	$\mathcal{K}_{k(A,r_0)}$	$\mathcal{K}_{k(A^T,\tilde{r}_0)}$	Does not solve an optimization problem
QMR	general A	$\mathcal{K}_{k(A,r_0)}$	no $\mathcal{C}_k$	$x_k \approx \arg\min_{x_0+\mathcal{K}_{k(A,r_0)}} \|b - Ax\|_2$

[1] Symmetric positive definite A. $\forall x \neq 0,\ x^T Ax > 0$. [2] Vector 2-norm. $\|x\|_2 = \sqrt{x^T x}$.

3.3. The Arnoldi algorithm for orthonormal basis

If we form the matrix $K_j(A, r_0)$,

$$K_j(A, r_0) = [r_0,\ Ar_0,\ A^2 r_0,\ \cdots,\ A^{j-1} r_0] \tag{43}$$

because $A^j r_0$ converges to an eigenvector corresponding to the largest eigenvalue in magnitude, it is likely that $K_j(A, r_0)$ is very ill-conditioned for large j. In order to avoid numerical troubles, we want to find orthonormal vectors $\{v_i\}_{i=1}^j$ such that

$$\mathcal{K}_j(A, r_0) = \text{range}(K_j(A, r_0)) = \text{span}\{v_1, v_2, \cdots, v_j\} \tag{44}$$

The idea is to use the Gram–Schmidt orthogonalization process and we show one step of it in the following.

Suppose we want to find v_1 and v_2 such that

$$\mathcal{K}_2(A, r_0) = \text{span}\{v_1, v_2\},\ v_i^T v_j = \delta_{ij},\ i, j = 1, 2 \tag{45}$$

Then this can be done by:

1. Normalize v_1: $v_1 := r_0/\|r_0\|_2$, then $\|v_1\|_2 = 1$.
2. Let $w_1 := Av_1$, then
$$\mathcal{K}_2(A, r_0) = \text{span}\{v_1, w_1\} \tag{46}$$
3. Orthogonalize w_1 against v_1: $v_2 := w_1 - (w_1^T v_1)v_1$, then
$$v_2^T v_1 = w_1^T v_1 - (v_1^T v_1)(w_1^T v_1) = 0 \tag{47}$$
and we still have
$$\mathcal{K}_2(A, r_0) = \text{span}\{v_1, v_2\} \tag{48}$$
4. Normalize v_2: $v_2 := v_2/\|v_2\|_2$, then $\|v_2\|_2 = 1$ and we still have $\mathcal{K}_2(A, r_0) = \text{span}\{v_1, v_2\}$.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 29 June 2023 doi:10.20944/preprints202306.2088.v1

We can write Arnoldi algorithm in terms of matrix. Let $V_k = [v_1, \dots, v_k]$ whose columns are composed by the orthonormal vectors $v_1, \dots, v_k$. After k steps, the algorithm has [8]

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T, \ k = 1, 2, \dots \tag{49}$$

$$H_k = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1,k-1} & h_{1,k} \\ h_{21} & h_{22} & \cdots & h_{2,k-1} & h_{2,k} \\ & h_{32} & \cdots & h_{3,k-1} & h_{3,k} \\ & & \ddots & & \vdots \\ & & & h_{k,k-1} & h_{k,k} \end{bmatrix}$$

where H_k is the $k \times k$ unreduced upper Hessenberg matrix with $h_{ij} = w_j^T v_i, i = 1, \dots, j$ and $h_{j+1,j} = \|w_j\|_2$ and e_k is the k-th column of I_k. At the m-th iteration, we get $h_{m+1,m} = 0$. Then Arnoldi algorithm stops and we have

$$AV_m = V_m H_m, \quad V_m^T V_m = I_m \tag{50}$$

4. Connections

One direct and most common application of the Krylov subspace methods onto VFA is to construct basis functions [11]. Recall as (5) shows, we need to solve linear system

$$(I - \gamma P)v = r, \ 0 \leq \gamma \leq 1 \tag{51}$$

Suppose $I - \gamma P \in \mathbb{R}^{n \times n}$ is nonsingular, since P is a stochastic matrix and $0 \leq \gamma < 1$, we have $\sigma(\gamma P) < 1$. If we want to approximate $(I - \gamma P)^{-1}$, one way is to use the Neumann series expansion [13],

$$(I - \gamma P)^{-1} = \sum_{i=0}^{\infty} (\gamma P)^i \tag{52}$$

Then the solution of (5) lies in $\mathcal{K}_m(P, r)$, where m is the degree of the minimal polynomial of P, as what shows below,

$$v = (I - \gamma P)^{-1} r = \sum_{i=0}^{\infty} (\gamma P)^i r \in \text{span}\{r, Pr, P^2 r, \dots\} = \text{span}\{r, Pr, P^2 r, \dots, P^{m-1} r\} \tag{53}$$

Therefore, we can set our feature matrix as $\Phi = K_m(P, r)$.

In practice, we could first start with a random initial guess θ_0 and $K_1(P, r)$ and progressively add $P^i r$ into the basis function matrix.

Parr [10] shows that the basis that constructed by Krylov subspace is equivalent the Bellman Error Basis Functions (BEBFs) and Model Error Basis Functions (MEBFs) which respectively add the Bellman error vectors and model error vectors progressively into the current basis.

Senda [15] applied the Conjugate Gradient (CG) algorithm, the Bi-Conjugate Gradient (BiCG) algorithm and the Block Product-type Krylov Subspace Method (BPKSM) for policy evaluation and did a comparison with the existing stationary iterative methods. What he concluded was that the algorithms based on the Krylov subspace methods were tens to hundreds times more efficient than conventional algorithms. He found the convergence rate of Krylov subspace algorithms improved as the iteration number increased, whereas the other algorithms remained the same. Algorithms based on Krylov subspace methods are far more efficient than the classical methods. The Krylov subspace methods can also be used in other machine learning area [19–24].

 doi:10.20944/preprints202306.2088.v1

5. Conclusion and Discussion

From the last section we can see that, constructing the basis functions is only a simple and direct usage of Krylov subspace methods. It works better on model-based learning, in which we have enough information of the transition function P. But whether or not it can be extended to model-free learning has not been well studied. Bertsekas [2] provided a good example on how to connect proximal iterations for solving $x = Ax + b$ with TD(λ) methods and got some interesting results. We can get some inspiration from it and try to use Krylov subspace methods for linear systems to solve approximate DP problems as well.

References

1. Dimitri P. Bertsekas, Dynamic Programming and Stochastic Control lecture notes, 2015 fall, MIT.
2. Dimitri P. Bertsekas, Proximal algorithms and temporal differences for large linear systems: extrapolation, approximation, and simulation, Report LIDS-P-3205, MIT; arXiv preprint arXiv: 1610.05427, 2016.
3. Justin A. Boyan, Technical update: Least-squares temporal difference learning, Machine Learning, 49(2-3):233-246, 2002.
4. Steven J. Bradtke and Andrew G. Barto, Linear least-squares algorithms for temporal difference learning, Machine Learning, 22:33-57, 1996.
5. Gene H. Golub and Charles F. Van Loan, Matrix Computations, 4th Edn., The Johns Hopkins University Press, Baltimore, 2013, Chapter 11.
6. Michail G. Lagoudakis and Ronald Parr, Least-squares policy iteration, Journal of Machine Learning Research, 4:1107-1149, 2003.
7. Alessandro Lazaric, Mohammad Ghavamzadeh and Rémi Munos, Finite-sample analysis of LSTD, In Proceedings of the 27th International Conference on Machine Learning, 2010.
8. Jorg Liesen and Zdenek Strakos, Krylov Subspace Methods: Principles and Analysis, Oxford University Press, 2015, Chapter 1 and 2.
9. Angelia Nedić and Dimitri P. Bertsekas, Least-squares policy evaluation algorithms with linear function approximation, Discrete Event Dynamic Systems: Theory and Applications, 13(1-2):79-110, 2003.
10. Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield and Michael L. Littman, An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning, Proceedings of the 25th international conference on Machine learning, pp.752-759, Helsinki, Finland, July 05-09, 2008.
11. Marek Petrik, An analysis of Laplacian methods for value function approximation in MDPs, Proceedings of the 20th international joint conference on Artifical intelligence, pp.2574-2579, Hyderabad, India, January 06-12, 2007.
12. Pascal Poupart and Craig Boutilier, VDCBPI: an approximate scalable algorithm for large scale POMDPs, Advances in Neural Information Processing Systems 17 (NIPS-2004), pp.1081-1088, MIT Press, 2005.
13. Martin L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley & Sons, Inc., New York, NY, 1994, Chapter 6.
14. Bruno Scherrer, Should one compute the temporal difference fixed point or minimize the Bellman residual: the unified oblique projection view, Proceedings of International Conference on Machine Learning, Haifa, Israel, pp.959-966, 2010.
15. K. Senda, S. Hattori, T. Hishinuma and T. Kohda, Acceleration of reinforcement learning by policy evaluation using nonstationary iterative method, IEEE Trans. Cybern., 44(12):2696-2705, Dec. 2014.
16. Csaba Szepesvári, Least squares temporal difference learning and Galerkin's method, Mini-Workshop: Mathematics of Machine Learning, Oberwolfach Reports, European Mathematical Society, Oberwolfach, pp. 2385-2388, 2011.
17. John N. Tsitsiklis and Benjamin Van Roy, An analysis of temporal-difference learning with function approximation, IEEE Trans. Automat. Contr., 42(5):674-690, 1997.
18. Huizhen Yu and Dimitri P. Bertsekas, Error bounds for approximations from projected linear equations, Mathematics of Operations Research, 35(2):306-329, 2010.

19. Sitao Luan, Mingde Zhao, Xiao-Wen Chang, Doina Precup (2019). Break the ceiling: Stronger multi-scale deep graph convolutional networks. Advances in neural information processing systems, 32.
20. Sitao Luan, Xiao-Wen Chang, Doina Precup (2019). Revisit Policy Optimization in Matrix Form. arXiv preprint arXiv:1909.09186.
21. Sitao Luan, Mingde Zhao, Chenqing Hua, Xiao-Wen Chang, Doina Precup (2020). Complete the missing half: Augmenting aggregation filtering with diversification for graph convolutional networks. arXiv preprint arXiv:2008.08844.
22. Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, Doina Precup (2021). Is Heterophily A Real Nightmare For Graph Neural Networks To Do Node Classification?. arXiv preprint arXiv:2109.05641.
23. Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, Doina Precup (2022). Revisiting heterophily for graph neural networks. arXiv preprint arXiv:2210.07606.
24. Sitao Luan, Chenqing Hua, Minkai Xu, Qincheng Lu, Jiaqi Zhu, Xiao-Wen Chang, Jie Fu, Jure Leskovec, Doina Precup (2023). When Do Graph Neural Networks Help with Node Classification: Investigating the Homophily Principle on Node Distinguishability. arXiv preprint arXiv:2304.14274.