Unrolling Dynamic Programming via Graph Filters

Sergio Rozada, Samuel Rey, Gonzalo Mateos, and Antonio G. Marques

Abstract—**Dynamic programming (DP) is a fundamental tool used across many engineering fields. The main goal of DP is to solve Bellman's optimality equations for a given Markov decision process (MDP). Standard methods like policy iteration exploit the fixed-point nature of these equations to solve them iteratively. However, these algorithms can be computationally expensive when the state-action space is large or when the problem involves long-term dependencies. Here we propose a new approach that unrolls and truncates policy iterations into a learnable parametric model dubbed BellNet, which we train to minimize the so-termed Bellman error from random value function initializations. Viewing the transition probability matrix of the MDP as the adjacency of a weighted directed graph, we draw insights from graph signal processing to interpret (and compactly re-parameterize) BellNet as a cascade of nonlinear graph filters. This fresh look facilitates a concise, transferable, and unifying representation of policy and value iteration, with an explicit handle on complexity during inference. Preliminary experiments conducted in a grid-like environment demonstrate that BellNet can effectively approximate optimal policies in a fraction of the iterations required by classical methods.**

Index Terms—**Algorithm Unrolling, Dynamic Programming, Graph Signal Processing, Graph Filter, Policy Iteration.**

I. Introduction

Dynamic programming (DP), recognized for its effectiveness in numerous engineering applications [1], is frequently modeled as a Markov decision process (MDP) [2]. A central challenge in DP involves solving Bellman's equations (BEQs) to determine value functions (VFs), which represent cumulative long-term rewards. Since BEQs constitute fixed-point equations, DP commonly relies on iterative algorithms [2], [3], wherein state transitions naturally induce a directed (di)graph structure. Despite their effectiveness, these iterative methods face significant computational hurdles. The number of required iterations until convergence grows rapidly with the size of the state-action space, and even more so in long-horizon problems.

To address these challenges, this work leverages algorithm unrolling [4], [5] and graph signal processing (GSP) [6], [7] to develop novel *learnable* neural architectures that reduce the number of DP iterations. Unrolling techniques combine the interpretability of model-based algorithms with the flexibility of data-driven methods [4], [8]. In unrolling, iterative algorithms are truncated to a finite sequence of update steps, eliminating conventional iterative loops. This sequential structure can then be mapped into a parametric model where iterations become layers, enabling particular elements of each update step to be learned directly from data rather than being prescribed [9].

The iterative nature of DP makes it particularly well-suited for unrolling strategies. In our approach, we unroll the steps of policy and value iteration into a deep architecture termed BellNet, enabling a compact, data-driven alternative to traditional DP solvers. To design and customize BellNet, we draw on tools from GSP. Specifically, we observe that each step in policy iteration can be formulated as a polynomial of the transition probability matrix, followed by a nonlinearity. By interpreting the transition matrix as the adjacency of a weighted digraph, we identify this matrix polynomial as a graph filter [10]. All in all, we find that unrolled policy iteration boils down to a cascade of nonlinear graph filters. This GSP crossover enables the design of encompassing unrolled architectures that (i) require fewer (inference) steps to approximate policy iteration; and (ii) transfer across similar environments. The summary of our contributions are

C1 We introduce BellNet, an unrolled version of policy iteration structured as a cascade of nonlinear graph filters;
C2 We put forth a learning problem, where the filter coefficients are trained to minimize the so-termed Bellman error from random VF initializations; and
C3 We experimentally show in a grid-world setting that the learned BellNet model converges in significantly fewer iterations and generalizes well to similar environments.

Prior work. In reinforcement learning (RL), unrolling has been used in image-based settings [11], and to learn the MDP topology by interpreting the transition matrix as a graph [12], [13]. Unlike our work, existing approaches (a) focus on value iteration, a special case of the more general policy iteration framework; (b) address RL rather than DP, thus they estimate transition probabilities instead of exploiting the graph structure to design the unrolled architecture; and (c) target single tasks instead of enabling generalization across MDPs.

Prior RL works have used GSP tools to improve algorithmic efficiency. For instance, [14] postulates the VFs lie in a low-dimensional subspace induced by the state transition digraph; [15] estimates the optimal policy on a subset of states and extends it via graph interpolation; and [16] applies graph reduction to simplify the decision process. While effective, these methods are task-specific. In contrast, BellNet is task-agnostic and applicable across different MDPs.

Finally, a growing body of work in GSP investigates the properties of graph filters, e.g., permutation equivariance, stability, or transferability [17]–[22]. We empirically show that BellNet inherits some of these desirable properties, although a deeper theoretical analysis is left for future work.

S. Rozada, S. Rey, and A. G. Marques are with the Dep. of Signal Theory, King Juan Carlos University, Madrid, Spain, {sergio.rozada, samuel.rey.escudero, antonio.garcia.marques}@urjc.es. G. Mateos is with the Dep. of ECE, University of Rochester, USA, gmateosb@ur.rochester.edu.

Work partially funded by the Spanish AEI (10.13039/501100011033) grant PID2022-136887NB-I00, and the Community of Madrid via the Ellis Madrid Unit and grants URJC/CAM F861 and F1180 and TEC-2024/COM-89. Minor edits of this document were made with the assistance of ChatGPT.

II. Preliminaries: Fundamentals of DP and GSP

DP. In DP, we consider an MDP defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R})$, where $\mathcal{S}$ and $\mathcal{A}$ are discrete state and action spaces, $\mathbf{P} \in [0,1]^{|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|}$ is a known transition probability matrix whose rows, indexed by state-action pairs (s,a), define distributions over next states s', and $\mathbf{R} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$ contains the rewards. Solving the MDP amounts to finding a policy $\pi : \mathcal{S} \mapsto [0,1]^{|\mathcal{A}|}$ that maximizes the VFs, defined as expected cumulative rewards. A policy maps each state s to a distribution over actions a, and the VF under π is given by $Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$, where $\gamma \in [0,1)$ is a discount factor and the instantaneous reward r_t is the entry of $\mathbf{R}$ indexed by the state-action pair at time t. We arrange policy probabilities in the matrix $\mathbf{\Pi} \in [0,1]^{|\mathcal{S}| \times |\mathcal{A}|}$ and the VFs in $\mathbf{Q}_{\pi} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$. For convenience, we henceforth use the vectorizations $\mathbf{r} = \mathrm{vec}(\mathbf{R})$ and $\mathbf{q}_{\pi} = \mathrm{vec}(\mathbf{Q}_{\pi})$.

The BEQs characterize the VFs $\mathbf{q}_{\pi}$ for a fixed policy π [23]. Denoting $\mathbf{P}_{\pi} = \mathbf{P}(\mathbf{I} \odot \mathbf{\Pi}^{\top})^{\top}$, where $\odot$ is the Khatri-Rao product and $\mathbf{I}$ the identity matrix, we have that

$$\mathbf{q}_{\pi} = \mathbf{r} + \gamma \mathbf{P}_{\pi} \mathbf{q}_{\pi}. \tag{1}$$

This fixed-point linear system of equations can be solved iteratively. Iterating until convergence is referred to as *policy evaluation* in DP parlance. Greedy maximization of $\mathbf{Q}_{\pi}$ with respect to actions (columns) produces a new policy $\mathbf{\Pi}'$, i.e.,

$$\Pi'_{i,j} = \begin{cases} 1 & \text{if } j = \arg\max_k Q_{ik}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

This step, known as *policy improvement*, produces a policy $\mathbf{\Pi}'$ that is guaranteed to outperform $\mathbf{\Pi}$ in terms of the attained VFs [3]. Crucially, if $\mathbf{\Pi}' = \mathbf{\Pi}$, then $\mathbf{\Pi} = \mathbf{\Pi}^{\star}$ is optimal, i.e., attains the maximum VFs $\mathbf{Q}_{\pi} = \mathbf{Q}^{\star}$ for all state-action pairs. This process underpins *policy iteration*, an iterative method that alternates between policy evaluation and policy improvement to compute the optimal VFs and policy.

Interestingly, for the optimal VFs $\mathbf{Q}^{\star}$, it also holds that

$$\mathbf{q}^{\star} = \mathbf{r} + \gamma \mathbf{P} \mathbf{v}^{\star} \quad \text{with} \quad v_i^{\star} = \max_k Q_{ik}^{\star}. \tag{3}$$

This defines a nonlinear fixed-point system that can be solved iteratively through a procedure known as *value iteration*. Value iteration is equivalent to performing one step of the policy evaluation iteration followed by policy improvement [23].

GSP. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of N nodes $\mathcal{V}$ and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The connectivity of $\mathcal{G}$ is captured by the sparse adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where $A_{ij} \neq 0$ if and only if $(i,j) \in \mathcal{E}$, and the entry A_{ij} denotes the weight of the edge from node i to node j. A *graph signal* is a function defined on the set of nodes, represented as a vector $\mathbf{x} \in \mathbb{R}^N$, where x_i denotes the signal value at node i.

Graph filters are linear, topology-aware operators that process graph signals. They can be expressed as matrix polynomials of the adjacency matrix $\mathbf{A}$ [10], [24], namely

$$\mathbf{H} = \textstyle\sum_{j=0}^{N-1} h_j \mathbf{A}^j, \tag{4}$$

where $\mathbf{h} = [h_0, \ldots, h_{N-1}]^{\top}$ is the vector of filter coefficients. Since each power $\mathbf{A}^j$ encodes information about the j-hop neighborhood of $\mathcal{G}$, the output $\mathbf{y} = \mathbf{H}\mathbf{x}$ can be interpreted as a diffusion (or aggregation) of the input signal $\mathbf{x}$ across neighborhoods of increasing size, with the coefficients h_j weighting the contribution from each j-hop component [25].

III. Unrolling DP via GSP

Algorithm unrolling is a foundational technique for infusing model-based inductive bias into data-driven learning [8]. Given an iterative algorithm, unrolling builds a parametric mapping, typically a neural network, by assigning each iteration to a corresponding block, such as a network layer. The operations of the original algorithm are preserved and reinterpreted as layer-wise computations, enabling the model to learn algorithm-specific behavior from data. Next, we unroll policy iteration and draw GSP connections in the process. Each unrolled block consists of two main steps: policy evaluation, which involves solving (1), and policy improvement, where (2) is applied.

Policy evaluation. The BEQ (1) is a linear system of equations. However, solving it directly is often impractical due to the large size of state-action spaces. Instead, one typically iterates by applying the right-hand-side (rhs) of (1) repeatedly until convergence is reached. Additional simplifications exploit structural properties of the MDP, such as linear dynamics [26], [27], low-rank structure [28]–[30], or kernel-based representations [31], [32]. Here, instead, we propose leveraging the graph structure of the MDP. To elucidate this connection, we expand the BEQ recursion as follows

$$\begin{aligned} \mathbf{q}^{(k)} &= \mathbf{r} + \gamma \mathbf{P}_{\pi} \mathbf{q}^{(k-1)} = \mathbf{r} + \gamma \mathbf{P}_{\pi} \mathbf{r} + \gamma^2 \left(\mathbf{P}_{\pi}\right)^2 \mathbf{q}^{(k-2)} \\ &= \ldots = \textstyle\sum_{j=0}^{k-1} \gamma^j \left(\mathbf{P}_{\pi}\right)^j \mathbf{r} + \gamma^k \left(\mathbf{P}_{\pi}\right)^k \mathbf{q}^{(0)}. \end{aligned} \tag{5}$$

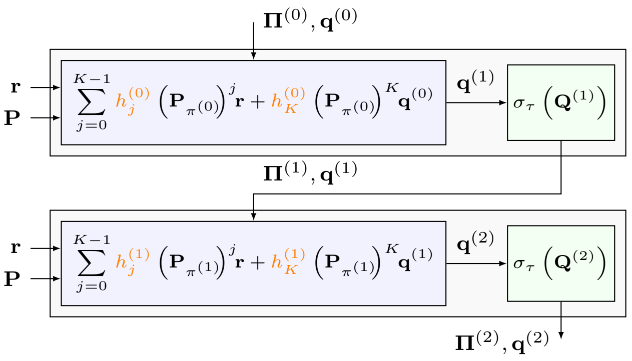
This expression has two terms: an exponentially decaying bias $\mathbf{b}^{(k)} := \gamma^k \left(\mathbf{P}_{\pi}\right)^k \mathbf{q}^{(0)}$ that depends on the initial value $\mathbf{q}^{(0)}$; and a *graph filter* $\mathbf{H}^{(k)} := \sum_{j=0}^{k-1} \gamma^j \left(\mathbf{P}_{\pi}\right)^j$ applied to $\mathbf{r}$. The latter characterization follows since $\mathbf{H}^{(k)}$ is a polynomial of $\mathbf{P}_{\pi}$, which represents the adjacency matrix of a weighted digraph $\mathcal{G}$. The nodes are state-action pairs while the edge weights correspond to the Markovian transition probabilities $\mathbf{P}$ and the current policy $\mathbf{\Pi}$. From this viewpoint, the powers of the discount factor γ act as the filter coefficients in (4), i.e., $h_j = \gamma^j$. Consequently, policy evaluation can be interpreted as applying a graph filter to the reward signal. Due to the fixed-point theorem [3], an infinite-order filter is guaranteed to recover the true VF for policy π, so that $\mathbf{q}_{\pi} = \mathbf{H}^{(\infty)} \mathbf{r} + \mathbf{b}^{(\infty)} = \sum_{j=0}^{\infty} \gamma^j \left(\mathbf{P}_{\pi}\right)^j \mathbf{r}$.

Moreover, our GSP perspective enables concrete simplifications of the proposed model. While the graph filter underlying policy evaluation is, in principle, of infinite degree, an equivalent filter with limited degree exists.

Proposition 1. *The value function $\mathbf{q}^{\pi}$ under a fixed policy π can be expressed as a finite-order graph filter*

$$\mathbf{q}_{\pi} = \textstyle\sum_{j=0}^{\infty} \gamma^j (\mathbf{P}_{\pi})^j \mathbf{r} = \sum_{j=0}^{K} \bar{h}_j (\mathbf{P}_{\pi})^j \mathbf{r}, \tag{6}$$

with $K \leq |\mathcal{S}||\mathcal{A}|$. If $\mathbf{P}_{\pi}$ is diagonalizable, then $K \leq |\mathcal{S}|$.



$\mathbf{\Pi}^{(0)}, \mathbf{q}^{(0)}$
$\mathbf{r}$
$\mathbf{P}$
$\sum_{j=0}^{K-1} h_j^{(0)} \left(\mathbf{P}_{\pi^{(0)}}\right)^j \mathbf{r} + h_K^{(0)} \left(\mathbf{P}_{\pi^{(0)}}\right)^K \mathbf{q}^{(0)}$
$\mathbf{q}^{(1)}$
$\sigma_\tau\left(\mathbf{Q}^{(1)}\right)$
$\mathbf{\Pi}^{(1)}, \mathbf{q}^{(1)}$
$\mathbf{r}$
$\mathbf{P}$
$\sum_{j=0}^{K-1} h_j^{(1)} \left(\mathbf{P}_{\pi^{(1)}}\right)^j \mathbf{r} + h_K^{(1)} \left(\mathbf{P}_{\pi^{(1)}}\right)^K \mathbf{q}^{(1)}$
$\mathbf{q}^{(2)}$
$\sigma_\tau\left(\mathbf{Q}^{(2)}\right)$
$\mathbf{\Pi}^{(2)}, \mathbf{q}^{(2)}$

Fig. 1: BellNet schematic. A cascade of learnable graph filters and row-wise softmax nonlinearities that unrolls policy iteration.

Proof. By the Cayley–Hamilton theorem [33], any matrix polynomial of $\mathbf{P}_\pi \in [0,1]^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}||\mathcal{A}|}$ can be reparameterized as a polynomial of degree at most $K = |\mathcal{S}||\mathcal{A}|$. If $\mathbf{P}_\pi$ is diagonalizable, the degree of its minimal polynomial is at most $\mathrm{rank}(\mathbf{P}_\pi) = \mathrm{rank}(\mathbf{P}) = |\mathcal{S}|$, so any polynomial of $\mathbf{P}_\pi$ can be expressed with order at most $K = |\mathcal{S}|$. □

Beyond exact policy evaluation, our approach also encompasses *approximate* policy evaluation via early stopping after a fixed number of iterations. Recall that value iteration corresponds to a *single* application of the rhs of (1). In any case, early stopping is equivalent to a graph filter of some order K and fixed coefficients $h_j = \gamma^j$. This also introduces a non-vanishing bias term that must be accounted for [cf. (5)]. Furthermore, the estimate $\hat{\mathbf{q}}_\pi$ may not converge to the true VF $\mathbf{q}_\pi$, so it must be reused to initialize the next policy evaluation under the updated policy $\mathbf{\Pi}'$. Identity (6) can be extended to this case by explicitly incorporating the bias term as

$$\hat{\mathbf{q}}_\pi = \textstyle\sum_{j=0}^{K-1} h_j (\mathbf{P}_\pi)^j \mathbf{r} + h_K \left(\mathbf{P}_\pi\right)^K \mathbf{q}^{(0)}. \qquad (7)$$

Policy improvement. As defined in (2), policy improvement is a nonlinear row-wise max operation applied to $\mathbf{Q}_\pi$, analogous to max-pooling, selecting the maximum in each row. For differentiability, we replace the max operation with a softmax, as detailed in the next section.

IV. BellNet: Learning Policy Iteration

Through the GSP lens, policy iteration is a cascade of nonlinear graph filtering operations that converge to the optimal VFs of the MDP. This perspective motivates BellNet, our proposed unrolling of policy iteration to solve BEQs. BellNet is a deep architecture composed of $L+1$ layers. Each layer takes as input a VF vector $\mathbf{q} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ and its associated softmax policy $\mathbf{\Pi} \in [0,1]^{|\mathcal{S}|\times|\mathcal{A}|}$, and outputs an enhanced VF vector and policy. The mapping between the input and output of the l-th layer is realized by a graph filter with learnable coefficients $\mathbf{h}^{(l)}$. Formally, let $\bar{\mathbf{q}}$ be the (possibly random) initial estimate of the VF, and let $\mathcal{H} = \{\mathbf{h}^{(l)}\}_{l=0}^{L}$ collect all the learnable coefficients. Then, BellNet, represented by the mapping $\mathbf{\Phi}(\cdot;\mathcal{H})$, implements $\{\hat{\mathbf{q}}, \hat{\mathbf{\Pi}}\} := \mathbf{\Phi}(\bar{\mathbf{q}};\mathcal{H})$ with $\hat{\mathbf{q}} = \mathbf{q}^{(L+1)}$, $\hat{\mathbf{\Pi}} = \mathbf{\Pi}^{(L+1)}$, $\mathbf{q}^{(0)} = \bar{\mathbf{q}}$, and layer-wise updates:

$$\mathbf{q}^{(l+1)} = \textstyle\sum_{j=0}^{K-1} h_j^{(l)} \left(\mathbf{P}_{\pi^{(l)}}\right)^j \mathbf{r} + h_K^{(l)} \left(\mathbf{P}_{\pi^{(l)}}\right)^K \mathbf{q}^{(l)}$$

$$\mathbf{\Pi}^{(l+1)} = \sigma_\tau(\mathbf{Q}^{(l+1)}), \text{ with } [\sigma_\tau(\mathbf{Q})]_{ij} = \frac{e^{Q_{ij}/\tau}}{\sum_{k=1}^{|\mathcal{A}|} e^{Q_{ik}/\tau}},$$

for $l = 0,\ldots,L$, where $\mathbf{Q}^{(l)} = \mathrm{unvec}(\mathbf{q}^{(l)})$, σ_τ is a row-wise softmax operator with temperature parameter τ, and $\mathbf{h}^{(l)} = [h_0^{(l)},\ldots,h_K^{(l)}]$ are the filter coefficients of the l-th layer. Each layer implements two reduced-order, parallel graph filters, sums their respective outputs, and then applies a softmax nonlinearity. The BellNet model is illustrated in Fig. 1. Notably, setting $L = \infty$ and $K = \infty$ with $h_j^{(l)} = \gamma^j$ and replacing the softmax with the max operator recovers policy iteration. Similarly, setting $K = 1$ yields value iteration.

Learning. To complete the approach, we formulate the optimization adopted to learn the filter coefficients $\mathcal{H}$. The loss function is inspired by temporal difference (TD) methods [34], [35]. We solve a sequential optimization problem that minimizes the Bellman error [36], [37], which is the discrepancy between the left and rhs of the optimal BEQ defined in (3). Specifically, with n being an iteration index, we solve

$$\mathcal{H}_{[n+1]} = \mathrm{argmin}_{\mathcal{H}} \|\mathbf{r} + \gamma \mathbf{P}_{\mathbf{\Pi}_{[n]}} \mathbf{q}_{[n]} - \mathbf{\Phi}(\bar{\mathbf{q}}, \mathcal{H})\|_2^2, \qquad (8)$$

where $\{\mathbf{q}_{[n]}, \mathbf{\Pi}_{[n]}\} := \mathbf{\Phi}(\bar{\mathbf{q}}, \mathcal{H}_{[n]})$. Note that $\{\mathbf{q}_{[n]}, \mathbf{\Pi}_{[n]}\}$ depends on the current iterate $\mathcal{H}_{[n]}$ and not on the optimized coefficients $\mathcal{H}$. By slight abuse of notation, $\mathbf{\Phi}$ in (8) refers only to the VF output $\hat{\mathbf{q}}$. We also highlight that: (a) as customary in TD, for each n we update the filter coefficients via gradient descent; (b) our DP method does not require data samples, but the transition probability matrix $\mathbf{P}$ instead; and (c) BellNet is initialized with an arbitrary VF $\bar{\mathbf{q}}$ and trained to converge to the optimal VF and policy regardless of $\bar{\mathbf{q}}$.

Transferability. Graph filters are permutation-equivariant and transferable to larger graphs from a convergent sequence [18], making them particularly well suited to generalize across related problems. In our DP context, this property can be leveraged to train BellNet on a single MDP and deploy it on other similar or larger MDPs. Doing so yields solutions faster than evaluating policies from scratch, as we demonstrate numerically in Section V. Moreover, the vanilla BellNet described so far operates with a fixed unrolling depth and distinct parameters per block. An attractive alternative is to *share* weights across blocks. Although weight sharing admittedly reduces expressiveness, it markedly decreases the number of learnable parameters [5], [38]. Crucially, it allows the same block to be reused as many times as desired during inference–exceeding the original training depth to enable efficient, scalable transfer as well as to delineate favorable complexity versus policy approximation tradeoffs.

V. Numerical Results and Concluding Remarks

We assess the performance of BellNet in the cliff walking environment, a grid-world setup where the goal is to reach a target location in the minimum number of steps without falling

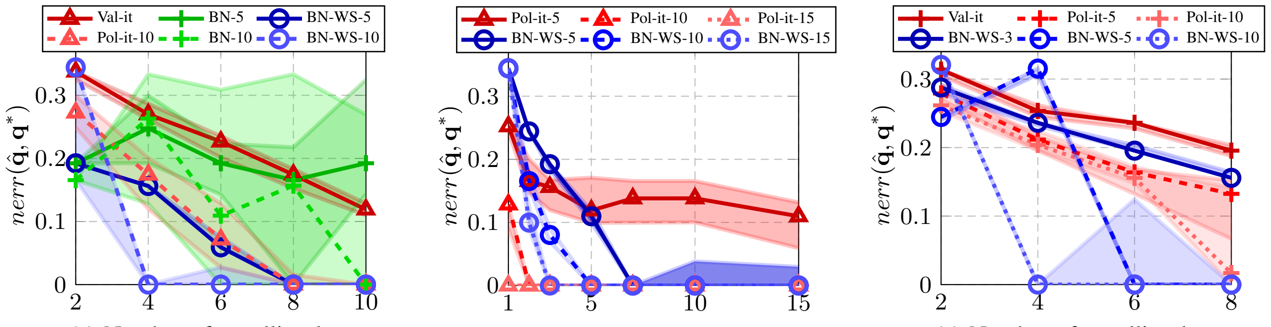


Fig. 2: Evaluation of BellNet across different scenarios. We report the median error of the estimated $\hat{\mathbf{q}}$, computed as in (9), over 15 realizations. a) Shows the error as L increases; b) illustrates the error as K increases; and c) evaluates the transferability capacity of BellNet.

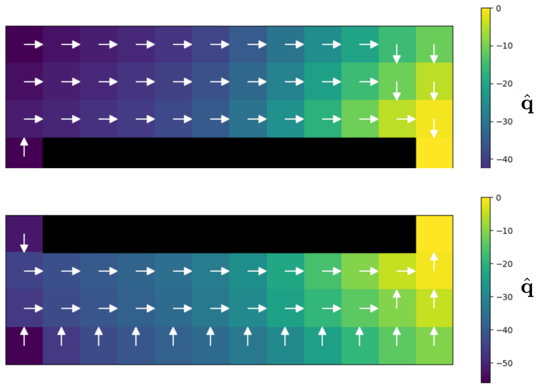


Fig. 3: Cliff walking environment (top) and its mirrored version (bottom). Cliff regions are shown in black; arrows indicate the policy learned by BellNet, and the color map represents the corresponding VFs. BellNet is trained on the top environment, while the policy in the bottom environment is inferred without retraining.

off the grid. The state space $\mathcal{S}$ corresponds to positions on the grid, and the action space consists of moving up, down, left, or right. Two instances of this environment are depicted in Fig. 3. Simulations are conducted using the Gymnasium library [39], [40], and the code is available on GitHub[1] for reproducibility.

We compute the true VF $\mathbf{q}^*$ using policy iteration with sufficiently many policy evaluation and improvement steps, and report the normalized error defined as

$$nerr(\hat{\mathbf{q}}, \mathbf{q}^*) = \left\|\hat{\mathbf{q}}/\|\hat{\mathbf{q}}\|_2 - \mathbf{q}^*/\|\mathbf{q}^*\|_2\right\|_2^2. \tag{9}$$

Figure 2 depicts the median error along with the interquartile range (between the 25th and 75th percentiles), computed over 15 random realizations. We compare the performance of BellNet with and without weight sharing (denoted “BN-WS” and “BN” in the legend), as well as value iteration (“Val-it”) and policy iteration (“Pol-it”), across multiple scenarios.

Test case 1 (Depth). We first examine how increasing the number of unrolling layers (equivalently, the number of policy improvement steps for “Val-it” and “Pol-it”) influences performance. Figure 2a shows results using filter orders 5 and 10 for “BN”, and 10 policy evaluation updates in “Pol-it”. Apparently, the weight sharing strategy leads to better performance with lower variance, whereas distinct filter coefficients results in more unstable behavior. Moreover, BellNet consistently outperforms policy iteration, recovering the optimal policy with only 4 layers compared to 10 required by “Pol-it”.

Test case 2 (Filter order). Next, we investigate the role of the filter order in the performance of BellNet. Figure 2b shows the error of “Pol-it” and “BN-WS” as the number of policy evaluation steps and, correspondingly, the filter order, increases as indicated on the x-axis. We consider 5, 10, and 15 policy improvement steps for “Pol-it” and the same number of unrolling layers for “BN-WS”. As expected, we find that a higher filter order improves the performance of “BN-WS”, with a smaller order being sufficient when the number of unrolling layers increases. Interestingly, this is not the case for “Pol-it”, where the number of policy improvement steps has a greater impact than the number of evaluation steps in this setting. Consistent with the previous experiment, these results highlight how BellNet outperforms “Pol-it” when the number of policy improvement steps is moderately small.

Test case 3 (Transferability). The last experiment inspects BellNet’s transferability properties. We train BellNet in the original grid-world setting used in previous test cases, and then use it to predict the optimal policy in a modified environment where the positions of the cliffs, origin, and destination have changed. As shown in Fig. 2c, BellNet successfully predicts the optimal policy in the new environment without requiring retraining. For comparison, we compute the optimal policy in the modified environment using value iteration and policy iteration with 5 and 10 policy evaluation steps. We observe that the error in the estimated $\hat{\mathbf{q}}$ decreases as the number of layers (indicated on the x-axis) increases, or, when higher-order filters are used. Notably, BellNet outperforms the classical baselines when both the number of unrolling layers and the filter order are sufficiently large. Overall, these preliminary results demonstrate that BellNet not only offers a novel approach to estimating the optimal policy, but also generalizes effectively to other related environments not seen during training.

[1]https://github.com/sergiorozada12/rl-unrolling

REFERENCES

[1] E. V. Denardo, *Dynamic Programming: Models and Applications*, Courier Corporation, 2012.

[2] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, 2014.

[3] D. Bertsekas, *Dynamic Programming and Optimal Control: Volume I*, vol. 4, Athena Scientific, 2012.

[4] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Int. Conf. Mach. Learning*, 2010, pp. 399–406.

[5] S. Chen, Y. C. Eldar, and L. Zhao, "Graph unrolling networks: Interpretable neural networks for graph signal denoising," *IEEE Trans. Signal Process.*, vol. 69, pp. 3699–3713, 2021.

[6] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[7] G. Leus, A. G. Marques, J. M. F. Moura, A. Ortega, and D. I. Shuman, "Graph signal processing: History, development, impact, and outlook," *IEEE Signal Process. Mag.*, vol. 40, no. 4, pp. 49–60, 2023.

[8] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, vol. 38, no. 2, pp. 18–44, 2021.

[9] S. Hadou, N. NaderiAlizadeh, and A. Ribeiro, "Robust stochastically-descending unrolled networks," *IEEE Trans. Signal Process.*, vol. 72, pp. 5484–5499, 2024.

[10] E. Isufi, F. Gama, D. I. Shuman, and S. Segarra, "Graph filters for signal processing and machine learning on graphs," *IEEE Trans. Signal Process.*, vol. 72, pp. 4745–4781, 2024.

[11] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Conf. Neural Inform. Process. Syst.*, 2016, vol. 29.

[12] S. Niu, S. Chen, H. Guo, C. Targonski, M. Smith, and J. Kovačević, "Generalized value iteration networks: Life beyond lattices," in *AAAI Conf. Artificial Intell.*, 2018, vol. 32.

[13] A. Deac, P.-L. Bacon, and J. Tang, "Graph neural induction of value iteration," *arXiv preprint arXiv:2009.12604*, 2020.

[14] L. Liu, A. Chattopadhyay, and U. Mitra, "On solving MDPs with large state space: Exploitation of policy structures and spectral properties," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4151–4165, 2019.

[15] L. Liu and U. Mitra, "Policy sampling and interpolation for wireless networks: A graph signal processing approach," in *Proc. IEEE Global Commun. Conf.*, 2019, pp. 1–6.

[16] M. Levorato, S. Narang, U. Mitra, and A. Ortega, "Reduced dimension policy iteration for wireless network control via multiscale analysis," in *Proc. IEEE Global Commun. Conf.*, 2012, pp. 3886–3892.

[17] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 5680–5695, 2020.

[18] L. Ruiz, F. Gama, and A. Ribeiro, "Graph neural networks: Architectures, stability, and transferability," *Proc. IEEE*, vol. 109, no. 5, pp. 660–682, 2021.

[19] R. Levie, W. Huang, L. Bucci, M. Bronstein, and G. Kutyniok, "Transferability of spectral graph convolutional neural networks," *J. Mach. Learning Res.*, vol. 22, no. 272, pp. 1–59, 2021.

[20] J. Cervino, L. Ruiz, and A. Ribeiro, "Learning by transference: Training graph neural networks on growing graphs," *IEEE Trans. Signal Process.*, vol. 71, pp. 233–247, 2023.

[21] S. Rey, M. Navarro, V. M. Tenorio, S. Segarra, and A. G. Marques, "Redesigning graph filter-based GNNs to relax the homophily assumption," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2025, pp. 1–5.

[22] Z. Wang, J. Cervino, and A. Ribeiro, "A manifold perspective on the statistical generalization of graph neural networks," in *Int. Conf. Learn. Representations*, 2025.

[23] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[24] S. Rey, V. M. Tenorio, and A. G. Marques, "Robust graph filter identification and graph denoising from signal observations," *IEEE Trans. Signal Process.*, vol. 71, pp. 3651–3666, 2023.

[25] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4117–4131, 2017.

[26] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learning Res.*, vol. 4, no. Dec, pp. 1107–1149, 2003.

[27] A. Geramifard et al., "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Foundations and Swerves® in Machine Learning*, vol. 6, no. 4, pp. 375–451, 2013.

[28] A. Agarwal, S. Kakade, A. Krishnamurthy, and W. Sun, "FLAMBE: Structural complexity and representation learning of low rank MDPs," in *Conf. Neural Inform. Process. Syst.*, 2020, vol. 33, pp. 20095–20107.

[29] S. Rozada, S. Paternain, and A. G. Marques, "Tensor and matrix low-rank value-function approximation in reinforcement learning," *IEEE Trans. Signal Process.*, vol. 72, pp. 1634–1649, 2024.

[30] S. Rozada, J. L. Orejuela, and A. G. Marques, "Solving finite-horizon MDPs via low-rank tensors," *arXiv preprint arXiv:2501.10598*, 2025.

[31] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Mach. Learn.*, vol. 49, no. 2, pp. 161–178, 2002.

[32] Y. Akiyama and K. Slavakis, "Nonparametric Bellman mappings for value iteration in distributed reinforcement learning," *arXiv preprint arXiv:2503.16192*, 2025.

[33] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, 2012.

[34] R. S. Sutton, *Reinforcement Learning: An Introduction*, A Bradford Book, 2018.

[35] M. Geist and O. Pietquin, "Algorithmic survey of parametric value function approximation," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 24, no. 6, pp. 845–867, 2013.

[36] D. Choi and B. Van Roy, "A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning," *Discrete Event Dyn. Syst.*, vol. 16, no. 2, pp. 207–239, 2006.

[37] K. Asadi, S. Sabach, Y. Liu, O. Gottesman, and R. Fakoor, "TD convergence: An optimization perspective," in *Conf. Neural Inform. Process. Syst.*, 2023, vol. 36, pp. 49169–49186.

[38] Steven J. Nowlan and Geoffrey E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural Comput.*, vol. 4, no. 4, pp. 473–493, 1992.

[39] G. Brockman et al., "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[40] M. Towers et al., "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.