The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Generalized Value Iteration Networks: Life Beyond Lattices

Sufeng Niu,[*†] **Siheng Chen,**[*§] **Hanyu Guo,**[*†]
Colin Targonski,[†] **Melissa C. Smith,**[†] **Jelena Kovačević**[‡]
[†] Clemson University, 433 Calhoun Dr., Clemson, SC 29634, USA
[‡] Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
[§] Uber Advanced Technologies Group, 100 32nd St, Pittsburgh, PA 15201, USA

Abstract

In this paper, we introduce a generalized value iteration network (GVIN), which is an end-to-end neural network planning module. GVIN emulates the value iteration algorithm by using a novel graph convolution operator, which enables GVIN to learn and plan on irregular spatial graphs. We propose three novel differentiable kernels as graph convolution operators and show that the embedding-based kernel achieves the best performance. Furthermore, we present episodic Q-learning, an improvement upon traditional n-step Q-learning that stabilizes training for VIN and GVIN. Lastly, we evaluate GVIN on planning problems in 2D mazes, irregular graphs, and real-world street networks, showing that GVIN generalizes well for both arbitrary graphs and unseen graphs of larger scale and outperforms a naive generalization of VIN (discretizing a spatial graph into a 2D image).

Introduction

Reinforcement learning (RL) is a technique that solves sequential decision making problems that lacks explicit rules and labels (Sutton and Barto 1998). Recent developments in Deep Reinforcement Learning(DRL) have lead to enormous progress in autonomous driving (Bojarski et al. 2016), innovation in robot control (Levine et al. 2015), and human-level performance in both Atari games (Mnih et al. 2013; Guo et al. 2014) and the board game Go (Silver et al. 2016a). Given a reinforcement learning task, the agent explores the underlying Markov Decision Process (MDP) (Bellman 1957; Bertsekas et al. 1995) and attempts to learn a mapping of high-dimensional state space data to an optimal policy that maximizes the expected return. Reinforcement learning can be categorized as model-free (Lillicrap et al. 2015; Mnih et al. 2016; 2013) and model-based approaches (Sutton and Barto 1998; Deisenroth and Rasmussen 2011; Schmidhuber 1990). Model-free approaches learn the policy directly by trial-and-error and attempt to avoid bias caused by a suboptimal environment model (Sutton and Barto 1998); the majority of recent architectures for DRL follow the model-free approach (Lillicrap et al. 2015; Mnih et al. 2016; 2013). Model-based approaches, on the other hand, allow for an agent to explicitly learn the mechanisms of an environment, which can lead to strong generalization abilities. A recent work, the value iteration networks (VIN) (Tamar et al. 2016) combines recurrent convolutional neural networks and max-pooling to emulate the process of value iteration (Bellman 1957; Bertsekas et al. 1995). As VIN learns an environment, it can plan shortest paths for unseen mazes.

The input data fed into deep learning systems is usually associated with regular structures. For example, speech signals and natural language have an underlying 1D sequential structure; images have an underlying 2D lattice structure. To take advantage of this regularly structured data, deep learning uses a series of basic operations defined for the regular domain, such as convolution and uniform pooling. However, not all data is contained in regular structures. In urban science, traffic information is associated with road networks; in neuroscience, brain activity is associated with brain connectivity networks; in social sciences, users' profile information is associated with social networks. To learn from data with irregular structure, some recent works have extended the lattice structure to general graphs (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2016) and redefined convolution and pooling operations on graphs; however, most works only evaluate data that has both a fixed and given graph. In addition, most lack the ability to generalize to new, unseen environments.

In this paper, we aim to enable an agent to self-learn and plan the optimal path in new, unseen spatial graphs by using model-based DRL and graph-based techniques. This task is relevant to many real-world applications, such as route planning of self-driving cars and web crawling/navigation. The proposed method is more general than classical DRL, extending for irregular structures. Furthermore, the proposed method is scalable (computational complexity is proportional to the number of edges in the testing graph), handles various edge weight settings and adaptively learns the environment model. Note that the optimal path can be self-defined, and is not necessarily the shortest one. Additionally, the proposed work differs from conventional planning algorithms; for example, Dijkstra's algorithm requires a known model, while GVIN aims to learn a general model via trial and error, then

[*] Equal contribution.

apply said model to new, unseen irregular graphs.

To create GVIN, we generalize VIN in two aspects. First, to work for irregular graphs, we propose a graph convolution operator that generalizes the original 2D convolution operator. With the new graph convolution operator, the proposed network captures the basic concepts in spatial graphs, such as direction, distance and edge weight. It also is able to transfer knowledge learned from one graph to others. Second, to improve reinforcement learning on irregular graphs, we propose a reinforcement learning algorithm, episodic Q-learning, which stabilizes the training for VIN and GVIN. The original VIN is trained through either imitation learning, which requires a large number of ground-truth labels, or reinforcement learning, whose performance is relatively poor. With the proposed episodic Q-learning, the new network performs significantly better than VIN in the reinforcement learning mode. Since the proposed network generalizes the original VIN model, we call it the *generalized value iteration network (GVIN)*.

The main contributions of this paper are:

- The proposed architecture, GVIN, generalizes the VIN (Tamar et al. 2016) to handle both regular structures and irregular structures. GVIN offers an end-to-end architecture trained via reinforcement learning (no ground-truth labels); see Section *Framework*;
- The proposed graph convolution operator generalizes 2D convolution learns the concepts of direction and distance, which enables GVIN to transfer knowledge from one graph to another; see Section *Graph Convolution*;
- The proposed reinforcement learning algorithm, episodic Q-learning, extends the classical n-step Q-learning as Monte Carlo control and significantly improves the performance of reinforcement learning for irregular graphs; see Section *Training via Reinforcement Learning*; and
- Through intensive experiments we demonstrate the generalization ability of GVIN within imitation learning and episodic Q-learning for various datasets, including synthetic 2D maze data, irregular graphs, and real-world maps (Minnesota highway and New York street maps); we show that GVIN significantly outperforms VIN with discretization input on irregular structures; See Section *Experimental Results*.

Background

Markov Decision Process. We consider an environment defined as an MDP that contains a set of states $s \in S$, a set of actions $a \in A$, a reward function $\mathbf{R}_{s,a}$, and a series of transition probabilities $\mathbf{P}_{s',s,a}$, the probability of moving from the current state s to the next state s' given an action a. The goal of an MDP is to find a policy that maximizes the expected return (accumulated rewards) $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where r_{t+k} is the immediate reward at the $(t+k)$th time stamp and $\gamma \in (0, 1]$ is the discount rate. A policy $\pi_{a,s}$ is the probability of taking action a when in state s. The value of state s under a policy π, $\mathbf{v}_s^\pi$, is the expected return when starting in s and following π; that is, $\mathbf{v}_s^\pi = \mathbb{E}[R_t | S_t = s]$. The value of taking action a in state s under a policy π, $\mathbf{q}^{\pi}{}_s^{(a)}$, is the expected return when starting in s, taking the action a and following π; that is, $\mathbf{q}^{\pi}{}_s^{(a)} = \mathbb{E}[R_t | S_t = s, A_t = a]$. There is at least one policy that is better than or equal to all other policies, called an optimal policy π^*; that is, the optimal policy is $\pi^* = \arg\max_\pi \mathbf{v}_s^\pi$, the optimal state-value function is $\mathbf{v}_s^* = \max_\pi \mathbf{v}_s^\pi$, and the optimal action-value function is $\mathbf{q}^{*}{}_s^{(a)} = \max_\pi \mathbf{q}^{\pi}{}_s^{(a)}$. To obtain π^* and $\mathbf{v}^*$, we usually consider solving the Bellman equation. Value iteration is a popular algorithm used to solve the Bellman equation in the discrete state space; that is, we iteratively compute $\mathbf{v}_s \leftarrow \max_a \sum_{s'} \mathbf{P}_{s',s,a} \left(\mathbf{R}_{s,a} + \gamma \mathbf{v}_{s'} \right)$ until convergence.

Differentiable planning module. VIN employs an embedded differentiable planning architecture, trained end-to-end via imitation learning (Tamar et al. 2016). In VIN, the Bellman equation is encoded within the convolutional neural networks, and the policy can be obtained through backpropagation. However, VIN is limited to regular lattices; it requires imitation learning for maximum performance and is trained separately with a reactive policy. A more recent work Memory Augmented Control Network (MACN) (Khan et al. 2017) combines the VIN model with a memory augmented controller, which can then backtrack through the history of previous trajectories. However, as we shown later in Table 2, GVIN outperform MACN on both performance and problem scales. A different model-based work, Predictron, uses a learning and planning model that simulates a Markov reward process (Silver et al. 2016b). The architecture unrolls the "imagined" plan via a predictron core. However, Predictron is limited to the Markov rewards process and is relatively computationally expensive compared to VIN.

Deep Learning with Graphs. A number of recent works consider using neural networks to handle signals supported on graphs (Niepert, Ahmed, and Kutzkov 2016; Duvenaud et al. 2015; Henaff, Bruna, and LeCun 2015). One principal idea is to generalize basic operations in the regular domain, such as filtering and pooling, to the graph domain based on spectral graph theory. For example, (Bruna et al. 2013; Henaff, Bruna, and LeCun 2015) introduce hierarchical clustering on graphs and the spectrum of the graph Laplacian to neural networks; (Defferrard, Bresson, and Vandergheynst 2016) generalizes classical convolutional neural networks by using graph coarsening and localized convolutional graph filtering; However, using these spectral-based approach cannot transfer the learned parameters from one graph to another, and thereby it cannot be used to plan paths in an unseen graph. On the other hand, vertex-based approach aims to learn the embedding of each node, which could generalize to unseen graph. For example, (Kipf and Welling 2016) considers semi-supervised learning with graphs by using graph-based convolutional neural networks; (Li et al. 2015) investigate learning graph structure through gated recurrent unit; (Gilmer et al. 2017) considers a message passing framework that unifies previous work. see overviews in (Bronstein et al. 2016).

Methodology

We propose a new model-based DRL framework, GVIN, that takes a general graph with a starting node and a goal node as inputs and outputs the designed plan. The goal of GVIN is to learn an underlying MDP that summarizes the optimal planning policy applied for arbitrary graphs, which

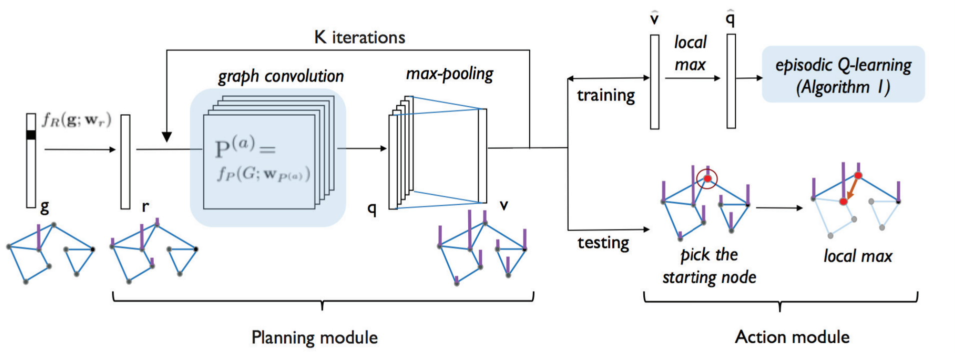


Figure 1: Architecture of GVIN. The left module emulates value iteration and obtains the state values; the right module is responsible for selecting an action based on an ϵ-greedy policy (for training) or a greed policy (for testing). We emphasize our contributions, including graph convolution operator and episodic Q-learning, in the blue blocks.

requires GVIN to capture general knowledge about planning that is structure and transition invariant and does not depend on any specific graph structure. A key component of an MDP is the transition matrix, which is needed to solve the Bellman equation. To train a general transition matrix that works for arbitrary graphs, similar to the VIN, we treat it as a graph convolution operator and parameterize it by using graph-based kernel functions, each of which represents a unique action pattern. We train the parameters in GVIN by using episodic Q-learning, which makes reinforcement learning on irregular graphs practical.

Framework

The input of GVIN is a graph with a starting node and a goal node. In the training phase, GVIN trains the parameters by trial-and-error on various graphs; during the testing phase, GVIN plans the optimal path based on the trained parameters. The framework includes the planning module (left) and the action module (right), shown in Figure 1. The planning module emulates value iteration by iteratively operating the graph convolution and max-pooling. The action module takes the greedy action according to the value function.

Mathematically, we consider a directed, weighted spatial graph $G = (\mathcal{V}, \mathbf{X}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V} = \{v_1, ..., v_N\}$ is the node set, $\mathbf{X} \in \mathbb{R}^{N\times 2}$ are the node embeddings with the ith row $\mathbf{X}_i \in \mathbb{R}^2$ being the embedding of the ith node (here we consider 2D spatial graphs, but the method is generalizable), $\mathcal{E} = \{e_1, ..., e_M\}$ is the edge set, and $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix, with the (i, j)th element $\mathbf{A}_{i,j}$ representing the edge weight between the ith and jth nodes. We consider a *graph signal* as a mapping from the nodes to real values. We use a graph signal $\mathbf{g} \in \{0, 1\}^N$ to encode the goal node, where $\mathbf{g}$ is one-sparse and only activates the goal node. Let $\mathbf{r} \in \mathbb{R}^N$, $\mathbf{v} \in \mathbb{R}^N$, and $\mathbf{q} \in \mathbb{R}^N$ be the reward graph signal, the state-value graph signal, and the action-value graph signal, respectively. We represent the entire process in a matrix-vector form as follows,

$$\mathbf{r} = f_R(\mathbf{g}; \mathbf{w_r}), \quad (1)$$

$$\mathbf{P}^{(a)} = f_P(G; \mathbf{w}_{\mathbf{P}^{(a)}}), \quad (2)$$

$$\mathbf{q}_{n+1}^{(a)} = \mathbf{P}^{(a)} \left(\mathbf{r} + \gamma \mathbf{v}_n\right), \quad (3)$$

$$\mathbf{v}_{n+1} = \max_a \mathbf{q}_{n+1}^{(a)}. \quad (4)$$

In the feature-extraction step (1), $\mathbf{g}$ is encoded to become the robust reward $\mathbf{r}$ via the feature-extract function $f_R(\cdot)$, which is a convolutional neural network in the case of regular graphs, but is the identity function when operating on irregular graphs; in step (2), where $\mathbf{P}^{(a)}$ is the graph convolution operator in the ath channel, a set of graph convolution operators is trained based on the graph G, which is further described in Section *Graph Convolution*; in (3) and (4), value iteration is emulated by using graph convolution to obtain the action-value graph signal $\mathbf{q}^{(a)}$ in the ath channel and max-pooling to obtain the state-value graph signal $\mathbf{v}$. $\mathbf{w_r}$ and $\mathbf{w}_{\mathbf{P}^{(a)}}$ are training parameters to parameterize $\mathbf{r}$ and $\mathbf{P}^{(a)}$, respectively. As shown in Figure 1, we repeat the graph convolution operation (3) and max-pooling (4) for K iterations to obtain the final state-value graph signal $\widehat{\mathbf{v}}$. When G is a 2D lattice, the planning module of GVIN degenerates to VIN.

In the training phase, we feed the final state-value graph signal $\widehat{\mathbf{v}}$ to the action module. The original VIN extracts the action values from step (3) and trains the final action probabilities for eight directions; however, this is problematic for irregular graphs, as the number of actions (neighbors) at each node varies. To solve this, we consider converting $\widehat{\mathbf{v}}$ to a pseudo action-value graph signal, $\widehat{\mathbf{q}} \in \mathbb{R}^N$, whose sth element is $\widehat{\mathbf{q}}_s = \max_{s' \in \mathrm{Nei}(s)} \widehat{\mathbf{v}}_{s'}$, representing the action value moving from s to one of its neighbors. The advantages of this approach come from the following three aspects: (1) the final state value of each node is obtained by using the maximum action values across all the channels, which is robust to small variations; (2) the pseudo action-value graph signal considers a unique action for each node and does not depend on the number of actions; that is, at each node, the agent queries the state values of its neighbors and always moves to the one with the highest value; and (3) the pseudo action-value graph signal considers local graph structure, because the next state is always chosen from one of the neighbors of the current state.

The pseudo action-value graph signal is used through episodic Q-learning, which learns from trial-and-error experience and backpropagates to update all of the training parameters. In episodic Q-learning, each episode is obtained as follows: for each given starting node s_0, the agent will move sequentially from s_t to s_{t+1} by the ϵ-greedy strategy; that is, with probability $(1-\epsilon)$, $s_{t+1} = \arg\max_{s' \in \text{Nei}(s_t)} \widehat{\mathbf{v}}_{s'}$ and with probability ϵ, s_{t+1} is randomly selected from one of the neighbors of s_t. An episode terminates when s_{t+1} is the goal state or the maximum step threshold is reached. For each episode, we consider the loss function as, $L(\mathbf{w}) = \sum_{t=1}^{T} \left(R_t - \widehat{\mathbf{q}}_{s_t}\right)^2$, where $\widehat{\mathbf{q}}_{s_t}$ is a function of the training parameters $\mathbf{w} = [\mathbf{w_r}, \mathbf{w_{P^{(a)}}}]$ in GVIN, T is the episode length and R_t is the expected return at time stamp t, defined as $R_t = (r_{t+1} + \gamma R_{t+1})$, where γ is the discount factor, and r_t is the immediate return at time stamp t. Additional details of the algorithm will be discussed in Section *Training via Reinforcement Learning*. In the testing phase, we obtain the action by greedily selecting the maximal state value; that is, $s_{t+1} = \arg\max_{s' \in \text{Nei}(s_t)} \widehat{\mathbf{v}}_{s'}$.

Graph Convolution

The conventional CNN takes an image as input, which is a 2D lattice graph. Each node is a pixel and has the same local structure, sitting on a grid and connecting to its eight neighbors. In this case, the convolution operator is easy to obtain. In irregular graphs, however, nodes form diverse local structures, making it challenging to obtain a structured and translation invariant operator that transfers knowledge from one graph to another. The fundamental problem here is to find a convolution operator that works for arbitrary local structures. We solve this through learning a 2D spatial kernel function that provides a transition probability distribution in the 2D space, and according to which we evaluate the weight of each edge and obtain a graph convolution operator.

The 2D spatial kernel function assigns a value to each position in the 2D space, which reflects the possibility to transit to the corresponding position. Mathematically, the transition probability from a starting position $\mathbf{x} \in \mathbb{R}^2$ to another position $\mathbf{y} \in \mathbb{R}^2$ is $K(\mathbf{x}, \mathbf{y})$, where $K(\cdot, \cdot)$ is a 2D spatial kernel function, which will be specified later.

Definition 1. A 2D spatial kernel function $K(\cdot, \cdot)$ is shift invariant when it satisfies $K(\mathbf{x}, \mathbf{y}) = K(\mathbf{x}+\mathbf{t}, \mathbf{y}+\mathbf{t})$, for all $\mathbf{x}, \mathbf{y}, \mathbf{t} \in \mathbb{R}^2$.

The shift invariance requires that the transition probability depend on the relative position, which is the key for transfer learning; in other words, no matter where the starting position is, the transition probability distribution is invariant. Based on a shift-invariant 2D spatial kernel function and the graph adjacency matrix, we obtain the graph convolution operator $\mathbf{P} = f_P(G; \mathbf{w_P}) \in \mathbb{R}^{N \times N}$, where each element $\mathbf{P}_{i,j} = \mathbf{A}_{i,j} \cdot K_{\mathbf{w_P}}(\mathbf{X}_i, \mathbf{X}_j)$, where the kernel function $K_{\mathbf{w_P}}(\cdot, \cdot)$ is parameterized by $\mathbf{w_P}$ and $\mathbf{X}_i, \mathbf{X}_j \in \mathbb{R}^2$ are the embeddings of the ith and jth node. The graph convolution operator follows from (1) graph connectivity and (2) 2D spatial kernel function. With the shift-invariant property, the 2D spatial kernel function leads to the same local transition distribution at each node; the graph adjacency matrix works as a modulator to select activations in the graph convolution operator. When there is no edge between i and j, we have $\mathbf{A}_{i,j} = 0$ and $\mathbf{P}_{i,j} = 0$; when there is an edge between i and j, $\mathbf{P}_{i,j}$ is high when $K_{\mathbf{w_P}}(\mathbf{X}_i, \mathbf{X}_j)$ is high; in other words, when the transition probability from the ith node to the jth node is higher, the edge weight $\mathbf{P}_{i,j}$ is high and the influence from the ith node to the jth node is bigger during the graph convolution. Note that $\mathbf{P}$ is a sparse matrix and its sparsity pattern is the same with its corresponding adjacency matrix, which ensures cheap computation.

As shown in (2), the graph convolution is a matrix-vector multiplication between the graph convolution operator $\mathbf{P}$ and the graph signal $\mathbf{r} + \gamma \mathbf{v}_n$; see Figure 2. Note that when we work with a lattice graph and an appropriate kernel function, this graph convolution operator $\mathbf{P}$ is nothing but a matrix representation of the conventional convolution (LeCun, Bengio, and others); in other words, VIN is a special case of GVIN when the underlying graph is a 2D lattice; see more details in Supplementary.

We consider three types of shift-invariant 2D spatial kernel functions: the directional kernel, the spatial kernel, and the embedding kernel.

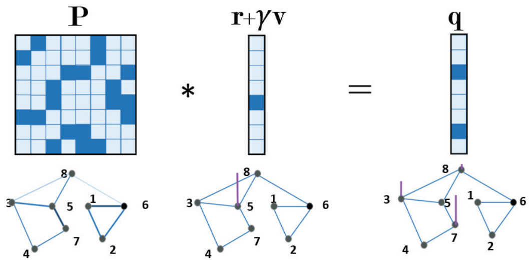


Figure 2: Matrix-vector multiplication as graph convolution. Through a graph convolution operator $\mathbf{P}$, $\mathbf{r} + \gamma \mathbf{v}$ diffuses over the graph to obtain the action-value graph signal $\mathbf{q}$.

Directional Kernel. The directional kernel is embedded with the direction information. The (i,j)th element in the graph convolution operator models the probability of following the edge from i to j ; that is,

$$\mathbf{P}_{i,j} \;=\; \mathbf{A}_{i,j} \cdot \sum_{\ell=1}^{L} w_\ell K_{\mathrm{d}}^{(t,\theta_\ell)}\left(\theta_{ij}\right), \tag{5}$$

$$\text{where } K_{\mathrm{d}}^{(t,\theta_\ell)}(\theta) \;=\; \left(\frac{1+\cos(\theta-\theta_\ell)}{2}\right)^t,$$

w_ℓ is kernel coefficient, θ_{ij} is the direction of the edge connecting the ith and the$j$th nodes, which can be computed through the node embeddings $\mathbf{X}_i, \mathbf{X}_j \in \mathbb{R}^2$, and $K_{\mathrm{d}}^{(t,\theta_\ell)}(\theta)$ is the directional kernel with order t and reference direction θ_ℓ, reflecting the center of the activation. The hyperparameters include the number of directional kernels L and the order

t, reflecting the directional resolution (a larger t indicates more focus in one direction). The kernel coefficient w_ℓ and the reference direction θ_ℓ are the training parameters, which is $\mathbf{w_P}$ in (2).

Spatial Kernel. We next consider both direction and distance. The (i,j)th element in the graph convolution operator is then,

$$\mathbf{P}_{i,j} \ = \ \mathbf{A}_{i,j} \cdot \sum_{\ell=1}^{L} w_\ell K_{\mathrm{s}}^{(d_\ell,t,\theta_\ell)}\left(d_{ij},\theta_{ij}\right),$$

$$\text{where } K_{\mathrm{s}}^{(d_\ell,t,\theta_\ell)}\left(d,\theta\right) \ = \ \mathrm{I}_{|d-d_\ell|\le\epsilon}\left(\frac{1+\cos(\theta-\theta_\ell)}{2}\right)^t, \tag{6}$$

and d_{ij} is the distance between the ith and the jth nodes, which can be computed through the node embeddings $\mathbf{X}_i, \mathbf{X}_j \in \mathbb{R}^2$, $K_{\mathrm{s}}^{(d_\ell,t,\theta_\ell)}\left(d,\theta\right)$ is the spatial kernel with reference distance d_ℓ and reference direction θ_ℓ and the indicator function $\mathrm{I}_{|d-d_\ell|\le\epsilon} = 1$ when $|d-d_\ell| \le \epsilon$ and 0, otherwise. The hyperparameters include the number of directional kernels L, the order t, the reference distance d_ℓ and the distance threshold ϵ. The kernel coefficient w_ℓ and the reference direction θ_ℓ are training parameters ($\mathbf{w_P}$ in (2)).

Embedding-based Kernel. In the directional kernel and spatial kernel, we manually design the kernel and provide hints for GVIN to learn useful direction-distance patterns. Now we directly feed the node embeddings and allow GVIN to automatically learn implicit hidden factors for general planning. The (i,j)th element in the graph convolution operator is then,

$$\mathbf{P}_{i,j} \ = \ \frac{(\mathrm{I}_{i=j} + \mathbf{A}_{i,j})}{\sqrt{\sum_k (1+\mathbf{A}_{k,j}) \sum_k (1+\mathbf{A}_{i,k})}} \cdot K_{\mathrm{emb}}\left(\mathbf{X}_i, \mathbf{X}_j\right), \tag{7}$$

where the indicator function $\mathrm{I}_{i=j} = 1$ when $i = j$, and 0, otherwise, and the embedding-based kernel function is $K_{\mathrm{emb}}\left(\mathbf{X}_i, \mathbf{X}_j\right) = \mathrm{mnnet}\left(\left[\mathbf{X}_i - \mathbf{X}_j\right]\right)$, with $\mathrm{mnnet}(\cdot)$ is a standard multi-layer neural network. The training parameters $\mathbf{w_P}$ in (2) are the weights in the multi-layer neural network. In practice, when the graph is weighted, we may also include the graph adjacency matrix $\mathbf{A}_{i,j}$ as the input of the multi-layer neural network.

Theorem 1. The proposed three kernel functions, the directional kernel, the spatial kernel and the embedding-based kernel, are shift invariant.

The proof follows from the fact that those kernels use only the direction, distance and the difference between two node embeddings, which only depend on the relative position.

Training via Reinforcement Learning

We train GVIN through episodic Q-learning, a modified version of n-step Q-learning. The difference between episodic Q-learning and the n-step Q-learning is that the n-step Q-learning has a fixed episode duration and updates the training weights after n steps; while in episodic Q-learning, each episodic terminates when the agent reaches the goal or the maximum step threshold is reached, and we update the trainable weights after the entire episode. During experiments, we found that for both regular and irregular graphs, the policy planned by the original Q-learning keeps changing and does not converge due to the frequent updates. Similar to the Monte Carlo algorithms (Sutton and Barto 1998), episodic Q-learning first selects actions by using its exploration policy until the goal is reached. Afterwards, we accumulate the gradients during the entire episode and then update the trainable weights, allowing the agent to use a stable plan to complete an entire episode. This simple change greatly improves the performance (see Section *Revisting 2D Mazes*). The pseudocode for the algorithm is presented in Supplementary.

Experimental Results

In this section, we evaluate the proposed method on three types of graphs: 2D mazes, synthesized irregular graphs and real road networks. We first validate that the proposed GVIN is comparable to the original VIN for 2D mazes, which have regular lattice structure. We next show that the proposed GVIN automatically learns the concepts of direction and distance in synthesized irregular graphs through the reinforcement learning setting (without using any ground-truth labels). Finally, we use the pre-trained GVIN model to plan paths for the Minnesota road network and Manhattan street network. Additional experiment parameter settings are listed in the Supplementary.

Revisting 2D Mazes

Given a starting point and a goal location, we consider planning the shortest paths for 2D mazes. We generate $22,467$ 2D mazes (16×16) using the same scripts[1] that VIN used. We use the same configuration as VIN ($6/7$ data for training and $1/7$ data for testing). Here we consider four comparisons: VIN vs. GVIN, action-value based imitating learning vs. state-value based imitating learning, direction-guided GVIN vs. unguided GVIN, and reinforcement learning.

Four metrics are used to quantify the planning performance, including *prediction accuracy*—the probability of taking the ground-truth action at each state (higher means better); *success rate*—the probability of successfully arriving at the goal from the start state without hitting any obstacles (higher means better); *path difference*—the average length difference between the predicted path and the ground-truth path (lower means better); and *expected reward*—the average accumulated reward (higher means better). The overall testing results are summarized in Table 1.

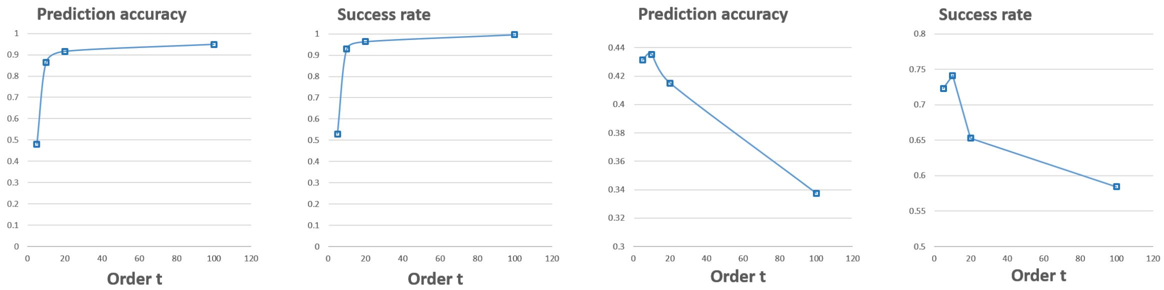
VIN vs. GVIN. GVIN performs competitively with VIN (Table 1), especially when GVIN uses direction-aware action-value based imitation learning (4th column in Table 1), which outperforms the others for all four metrics. The value map learned from GVIN with direction-unaware state-value based imitation learning is shown in Supplementary. We see negative values (in blue) at obstacles and positive values (in red) around the goal, which is similar to the value map that VIN reported in (Tamar et al. 2016).

Action-value vs. State-value. VIN with action-value imitation learning slightly outperforms VIN with state-value im-

[1] https://github.com/avivt/VIN

	VIN		GVIN			
	Action-value	**State-value**	**Action-value**		**State-value**	
			dir-aware	unaware	dir-aware	unaware
Prediction accuracy	95.00%	95.00%	**95.20%**	92.90%	94.40%	94.80%
Success rate	99.30%	99.78%	**99.91%**	98.60%	99.57%	99.68%
Path difference	0.089	0.010	**0.004**	0.019	0.013	0.015
Expected reward	0.963	0.962	**0.965**	0.939	0.958	0.960

Table 1: 2D Maze performance comparison for VIN and GVIN. GVIN achieves similar performance with VIN for 2D mazes (16×16); state-value imitation learning achieves similar performance with action-value imitation learning.



(a) Prediction accuracy in 2D mazes.

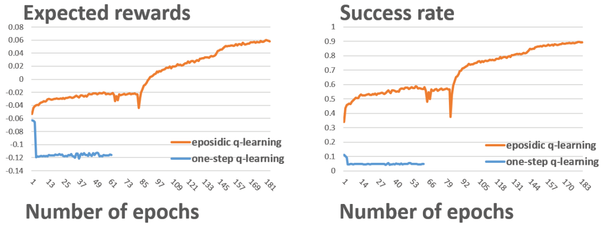
(b) Success rate in 2D mazes.

(c) Prediction accuracy in irregular graphs

(d) Success rate in irregular graphs.

Figure 3: Kernel direction order influences the planning performance in both regular and irregular graph.

itation learning. Similarly, GVIN with action-value based imitation learning slightly outperforms GVIN with state-value based imitation learning. The results suggest that our action approximation method (Section *Framework*) does not impact the performance while maintaining the ability to be extended to irregular graphs.



(a) Expected rewards.

(b) Success rate.

Figure 4: Q- vs. episodic Q-learning on 16×16 Maze.

Direction-aware GVIN vs. Unaware GVIN. Direction-aware GVIN slightly outperforms direction-unaware GVIN, which is reasonable because the fixed eight directions are ground truth for regular 2D mazes. It remains encouraging that the GVIN is able to find the ground-truth directions through imitation learning. As shown later, direction-unaware GVIN outperforms direction-aware GVIN in irregular graphs. Figures 3(a) and (b) show that the planning performance improves as the kernel exponential t in (5) increases due to the resolution in the reference direction being low when t is small. When $t = 5$, the kernel activates wide-range directions; when $t = 100$, the kernel focuses on a small-range directions and has a higher resolution.

Reinforcement Learning. We also examine the performance of episodic Q-learning (Section *Training via Reinforcement Learning*) in VIN. The Supplementary shows that the episodic Q-learning algorithm outperforms the training method used in VIN (TRPO + curriculum learning). For the results reported in Supplementary, we were able to train the VIN using our algorithm (episodic Q-learning) in just 200 epochs, while TRPO and curriculum learning took 1000 epochs to train VIN, as reported in (Tamar et al. 2016) (both algorithms used the same settings). As shown in Figure 4, the episodic Q-learning algorithm shows faster convergence and better overall performance when compared with Q-learning.

Exploring Irregular Graphs

We consider four comparisons in the following experiments: Directional kernel vs. Spatial kernel vs. Embedding-based kernel, direction-aware vs. direction-unaware, scale generalization, and reinforcement learning vs. imitation learning. We use the same performance metrics as the previously discussed 2D maze experiments.

Directional Kernel vs. Spatial Kernel vs. Embedding-based Kernel. We first train the GVIN via imitation learning. Table 2 shows that the embedding-based kernel outperforms the other kernel methods in terms of both action prediction and path difference (5th column in Table 2), indicating that the embedding-based kernel captures the edge weight information (distance) within the neural network weights better than the other methods. The spatial kernel demonstrates higher accuracy and success rate when compared with the directional kernel, which suggests the effectiveness of using

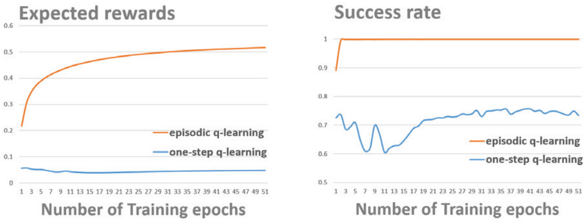
	VIN	**MACN**	**Directional Kernel**		**Spatial Kernel**		**Embedding-based Kernel**		
		(36 nodes)	dir-aware	unaware	dir-aware	unaware	100-IL[1]	10-IL[1]	10-RL[1]
Prediction acc.	26.57%	78%	41.50%	41.51%	57.45%	57.90%	**58.90%**	56.14%	50.90%
Success rate	10.29%	89.4%	34.75%	65.30%	96.56%	97.17%	97.34%	6.73%	**100%**
Path diff.	0.992	-	0.175	0.141	0.082	0.082	**0.079**	0.041	0.148
Expected reward	-0.905	-	0.266	0.599	0.911	0.917	0.922	-0.03	**0.943**

Table 2: The performance comparison among VIN and three different kernels of GVIN. All experiments except MACN (Khan et al. 2017) are tested on 100-node irregular graphs. The last column is trained using episodic Q-learning. IL and RL stands for imitate learning and reinforcement learning, respectively. Under similar experimental settings, MACN achieves an 89.4% success rate for 36-node graphs, while GVIN achieves a 97.34% success rate for 100-node graphs.

		Minnesota			**New York City**	
	Optimal	$\lvert\mathcal{V}\rvert = 100$	$\lvert\mathcal{V}\rvert = 10$	**Optimal**	$\lvert\mathcal{V}\rvert = 100$	$\lvert\mathcal{V}\rvert = 10$
Prediction Accuracy	100%	78.37%	78.15%	100%	78.66%	79.11%
Success rate	100%	100%	100%	100%	100%	100%
Path difference	0.0000	0.1069	0.1025	0.0000	0.03540	0.0353
Expected reward	0.96043	0.95063	0.95069	0.97279	0.97110	0.97136

Table 3: Performance comparison on Minnesota and New York City street map data using GVIN. $|\mathcal{V}| = 100$ is trained on 100-node graphs and $|\mathcal{V}| = 10$ is trained on 10-node graphs.

bin sampling. The *direction-unaware* method shows slightly better results for the spatial kernel, but has a larger success rate gain for the directional kernel. We also train VIN (1st column) by converting graph to 2D image. As shown in the Table, VIN fails significantly (See Supplementary).



(a) Expected rewards

(b) Success rate

Figure 5: Q- vs. Episodic Q-learning on irregular graphs.

Figures 3(c) and (d) show the planning performance for the irregular domain as the kernel order t in 5 increases. The results show that a larger t in the irregular domain has the opposite effect when compared with the regular domain. The observation is reasonable: in the irregular domain, the direction of each neighbor is extremely variable and a larger kernel order creates a narrower direction range, thus resulting in information loss.

Reinforcement Learning. We then train the GVIN using episodic Q-learning to compare with imitation learning. As a baseline, we also train GVIN by using standard deep Q-learning techniques, including using an experience replay buffer and a target network. Both networks use the same kernel function (embedding-based kernel) and configurations. Figure 5 shows the comparison of the two algorithms' success rate and expected rewards during the training. Clearly, episodic Q-learning converges to both a high success rate and a high expected rewards, but the standard deep Q-learning techniques fail to achieve reasonable results.

Scale Generalization. We also examine the scale generalization by training on 10-node graphs and then testing on 100-node graphs using the embedding-based kernel. When GVIN is trained on 10-node graphs via imitation learning, the performance is significantly hindered as shown in Table 2 (6th column). When GVIN is trained using episodic Q-learning, Table 2 (7th column) shows excellent generalization abilities that outperform all imitation learning based results for success rate and expected rewards. Compared with imitation learning, we also observe the performance decreases for path differences and action prediction.

Graph with Edge Weights. We also test how GVIN handles edge weights. We set the true weighted shortest path to be $\frac{\mathbf{X}_i - \mathbf{X}_j}{W_{ij}}$, where $\mathbf{X}_i - \mathbf{X}_j$ is the distance between two nodes and W_{ij} is the edge weight. As shown in Table in Supplementary, imitation learning is trained on 100-node graphs, while reinforcement learning is trained on 10-node. We also examine the GVIN by excluding edge weights from the input to see if there are any effects on performance.

Validating Real Road Networks

To demonstrate the generalization capabilities of GVIN, we evaluate two real-world maps: the Minnesota highway map, which contains 2642 nodes representing intersections and 6606 edges representing roads, and the New York City street map, which contains 5069 nodes representing intersections and 13368 edges representing roads (Chen et al.). We use the same models trained on the graphs containing $|\mathcal{V}| = 100$ and $|\mathcal{V}| = 10$ nodes with the embedding-based

[1] 100/10-IL stands for trained on 100/10 nodes with imitation learning respectively. 10-RL stands for trained on 10 nodes with reinforcement learning.

kernel and using episodic Q-learning in Section *Exploring Irregular Graphs*, separately. We normalize the data coordinates between 0 and 1, and we set recurrence parameter to $K = 200$. We randomly pick start points and goal points 1000 different times. We use the A* algorithm as a baseline. Table 3 shows that both $|\mathcal{V}| = 100$ and $|\mathcal{V}| = 10$ generalize well on large scale data. The policy could reach the goal position with 100% in the experiments. One sample planned path is shown in Supplementary.

Conclusions

We have introduced GVIN, a differentiable, novel planning module capable of both regular and irregular graph navigation and impressive scale generalization. We also introduced episodic Q-learning that is designed to stabilize the training process of VIN and GVIN. The proposed graph convolution may be applied to many other graph-based applications, such as navigation, 3D point cloud processing and molecular analysis, which is left for future works.

References

Bellman, R. 1957. Dynamic programming. *Princeton, USA: Princeton University Press* 1(2):3.

Bertsekas, D. P.; Bertsekas, D. P.; Bertsekas, D. P.; and Bertsekas, D. P. 1995. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA.

Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; and Vandergheynst, P. 2016. Geometric deep learning: going beyond euclidean data. *arXiv preprint arXiv:1611.08097*.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

Chen, S.; Yang, Y.; Kovacevic, J.; and Faloutsos, C. Monitoring manhattan's traffic from 5 cameras?

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 3837–3845.

Deisenroth, M., and Rasmussen, C. E. 2011. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 465–472.

Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, 2224–2232.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

Guo, X.; Singh, S.; Lee, H.; Lewis, R. L.; and Wang, X. 2014. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, 3338–3346.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

Khan, A.; Zhang, C.; Atanasov, N.; Karydis, K.; Kumar, V.; and Lee, D. D. 2017. Memory augmented control networks. *arXiv preprint arXiv:1709.05706*.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

LeCun, Y.; Bengio, Y.; et al. Convolutional networks for images, speech, and time series.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2015. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd annual international conference on machine learning. ACM*.

Schmidhuber, J. 1990. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, 253–258. IEEE.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016a. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D.; van Hasselt, H.; Hessel, M.; Schaul, T.; Guez, A.; Harley, T.; Dulac-Arnold, G.; Reichert, D.; Rabinowitz, N.; Barreto, A.; et al. 2016b. The predictron: End-to-end learning and planning. *arXiv preprint arXiv:1612.08810*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tamar, A.; Levine, S.; Abbeel, P.; WU, Y.; and Thomas, G. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2146–2154.