Value Iteration Networks

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel
Dept. of Electrical Engineering and Computer Sciences, UC Berkeley

Abstract
We introduce the value iteration network (VIN): a fully differentiable neural network with a ‘planning module’ embedded within. VINs can learn to plan, and are suitable for predicting outcomes that involve planning-based reasoning, such as policies for reinforcement learning. Key to our approach is a novel differentiable approximation of the value-iteration algorithm, which can be represented as a convolutional neural network, and trained end-to-end using standard backpropagation. We evaluate VIN based policies on discrete and continuous path-planning domains, and on a natural-language based search task. We show that by learning an explicit planning computation, VIN policies generalize better to new, unseen domains.

1 Introduction
Over the last decade, deep convolutional neural networks (CNNs) have revolutionized supervised learning for tasks such as object recognition, action recognition, and semantic segmentation [4, 15, 7, 19]. Recently, CNNs have been applied to reinforcement learning (RL) tasks with visual observations such as Atari games [21], robotic manipulation [18], and imitation learning (IL) [10]. In these tasks, a neural network (NN) is trained to represent a policy – a mapping from an observation of the system’s state to an action, with the goal of representing a control strategy that has good long-term behavior, typically quantified as the minimization of a sequence of time-dependent costs.

The sequential nature of decision making in RL is inherently different than the one-step decisions in supervised learning, and in general requires some form of planning [3]. However, most recent deep RL works [21, 18, 10] employed NN architectures that are very similar to the standard networks used in supervised learning tasks, which typically consist of CNNs for feature extraction, and fully connected layers that map the features to a probability distribution over actions. Such networks are inherently reactive, and in particular, lack explicit planning computation. The success of reactive policies in sequential problems is due to the learning algorithm, which essentially trains a reactive policy to select actions that have good long-term consequences in its training domain.

To understand why planning can nevertheless be an important ingredient in a policy, consider the grid-world navigation task depicted in Figure 1 (left), in which the agent can observe a map of its domain, and is required to navigate between some obstacles to a target position. One hopes that after training a policy to solve several instances of this problem with different obstacle configurations, the policy would generalize to solve a different, unseen domain, as in Figure 1 (right). However, as we show in our experiments, while standard CNN-based networks can be easily trained to solve a set of such maps, they do not generalize well to new tasks outside this set, because they do not understand the goal-directed nature of the behavior. This observation suggests that the computation learned by reactive policies is different from planning, which is required to solve a new task1.

1In principle, with enough training data that covers all possible task configurations, and a rich enough policy representation, a reactive policy can learn to map each task to its optimal policy. In practice, this is often too expensive, and we offer a more data-efficient approach by exploiting a flexible prior about the planning computation underlying the behavior.
In this work, we propose a NN-based policy that can effectively learn to plan. Our model, termed a value-iteration network (VIN), has a differentiable ‘planning program’ embedded within the NN structure.

The key to our approach is an observation that the classic value-iteration (VI) planning algorithm [2, 3] may be represented by a specific type of CNN. By embedding such a VI network module inside a standard feed-forward classification network, we obtain a NN model that can learn the parameters of a planning computation that yields useful predictions. The VI block is differentiable, and the whole network can be trained using standard backpropagation. This makes our policy simple to train using standard RL and IL algorithms, and straightforward to integrate with NNs for perception and control.

Our approach is different from model-based RL [25, 5], which requires system identification to map the observations to a dynamics model, which is then solved for a policy. In many applications, including robotic manipulation and locomotion, accurate system identification is difficult, and modelling errors can severely degrade the policy performance. In such domains, a model-free approach is often preferred [18]. Since a VIN is just a NN policy, it can be trained model free, without requiring explicit system identification. In addition, the effects of modelling errors in VINs can be mitigated by training the network end-to-end, similarly to the methods in [13, 12].

We demonstrate the effectiveness of VINs within standard RL and IL algorithms in various problems, among which require visual perception, continuous control, and also natural language based decision making in the WebNav challenge [23]. After training, the policy learns to map an observation to a planning computation relevant for the task, and generate action predictions based on the resulting plan. As we demonstrate, this leads to policies that generalize better to new, unseen, task instances.

2 Background

In this section we provide background on planning, value iteration, CNNs, and policy representations for RL and IL. In the sequel, we shall show that CNNs can implement a particular form of planning computation similar to the value iteration algorithm, which can then be used as a policy for RL or IL.

Value Iteration: A standard model for sequential decision making and planning is the Markov decision process (MDP) [2, 3]. An MDP $M$ consists of states $s \in S$, actions $a \in A$, a reward function $R(s, a)$, and a transition kernel $P(s'|s, a)$ that encodes the probability of the next state given the current state and action. A policy $\pi(a|s)$ prescribes an action distribution for each state. The goal in an MDP is to find a policy that obtains high rewards in the long term. Formally, the value $V^\pi(s)$ of a state under policy $\pi$ is the expected discounted sum of rewards when starting from that state and executing policy $\pi$, $V^\pi(s) \equiv \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s \right]$, where $\gamma \in (0, 1)$ is a discount factor, and $\mathbb{E}_\pi$ denotes an expectation over trajectories of states and actions $(s_0, a_0, s_1, a_1, \ldots)$, in which actions are selected according to $\pi$, and states evolve according to the transition kernel $P(s'|s, a)$. The optimal value function $V^*(s) = \max_\pi V^\pi(s)$ is the maximal long-term return possible from a state. A policy $\pi^*$ is said to be optimal if $V^{\pi^*}(s) = V^*(s) \ \forall s$. A popular algorithm for calculating $V^*$ and $\pi^*$ is value iteration (VI):

$$V_{n+1}(s) = \max_a Q_n(s, a) \quad \forall s, \quad \text{where} \quad Q_n(s, a) = R(s, a) + \gamma \sum_{s', a'} P(s'|s, a) V_n(s'). \quad (1)$$

It is well known that the value function $V_n$ in VI converges as $n \to \infty$ to $V^*$, from which an optimal policy may be derived as $\pi^*(s) = \arg \max_a Q_\infty(s, a)$.

Convolutional Neural Networks (CNNs) are NNs with a particular architecture that has proved useful for computer vision, among other domains [9, 16, 4, 15]. A CNN is comprised of stacked convolution and max-pooling layers. The input to each convolution layer is a 3-dimensional signal $X$, typically, an image with $l$ channels, $m$ horizontal pixels, and $n$ vertical pixels, and its output $h$ is a $l'$-channel convolution of the image with kernels $W^1, \ldots, W^l$, $h_{l', i', j'} = \sigma \left( \sum_{i,j} W_{l', i,j}^l X_{l, i+i', j+j'} \right)$, where $\sigma$ is some scalar activation function. A max-pooling layer selects, for each channel $l$ and pixel $i, j$ in $h$, the maximum value among its neighbors $N(i, j)$, $h_{l, i, j}^{\text{maxpool}} = \max_{i', j' \in N(i, j)} h_{l, i', j'}$. Typically, the neighbors $N(i, j)$ are chosen as a $k \times k$ image...
As stated earlier, the motivation for such a representation is that a natural solution to many tasks, such as weights. The goal is to tune the parameters such that the policy behaves well in the sense that \( \pi(s, a) \), only a subset of the input features (value function) is relevant. Attention is known to improve In NN terminology, this is a form of \( \bar{P} \) approach is based on two important observations. The first is that the vector of values \( M \) planning has the advantage of being differentiable, and simple to implement within a NN framework. For example, in the grid-world domain described above, we can let \( M \) on the observation in \( M \) planning has the advantage of being differentiable, and simple to implement within a NN framework. In this section however, we focus on how to use the planning result \( M \) is some unknown MDP Let \( M \) on the observation in \( M \) planning has the advantage of being differentiable, and simple to implement within a NN framework. In this section we introduce a general policy representation that embeds an explicit planning module. In RL, the optimal action is not available, but instead, the agent can act in the world and observe the rewards and state transitions its actions effect. RL algorithms such as in [27, 21, 26, 18] use these observations to improve the value of the policy.

3 The Value Iteration Network Model

In this section we introduce a general policy representation that embeds an explicit planning module. As stated earlier, the motivation for such a representation is that a natural solution to many tasks, such as the path planning described above, involves planning on some model of the domain.

Let \( M \) denote the MDP of the domain for which we design our policy \( \pi \). We assume that there is some unknown MDP \( \bar{M} \) such that the optimal plan in \( \bar{M} \) contains useful information about the optimal policy in the original task \( M \). However, we emphasize that we do not assume to know \( \bar{M} \) in advance. Our idea is to equip the policy with the ability to learn and solve \( M \), and to add the solution of \( M \) as an element in the policy \( \pi \). We hypothesize that this will lead to a policy that automatically learns a useful \( M \) to plan on. We denote by \( s \in S \), \( a \in A \), \( R(s, a) \), and \( P(s' | s, a) \) the states, actions, rewards, and transitions in \( \bar{M} \). To facilitate a connection between \( \bar{M} \) and \( M \), we let \( \bar{R} \) and \( \bar{P} \) depend on the observation in \( \bar{M} \), namely, \( \bar{R} = f_R(\bar{\phi}(s)) \) and \( \bar{P} = f_P(\bar{\phi}(s)) \), and we will later learn the functions \( f_R \) and \( f_P \) as a part of the policy learning process.

For example, in the grid-world domain described above, we can let \( \bar{M} \) have the same state and action spaces as the true grid-world \( M \). The reward function \( f_R \) can map an image of the domain to a high reward at the goal, and negative reward near an obstacle, while \( f_P \) can encode deterministic movements in the grid-world that do not depend on the observation. While these rewards and transitions are not necessarily the true rewards and transitions in the task, an optimal plan in \( M \) will still follow a trajectory that avoids obstacles and reaches the goal, similarly to the optimal plan in \( M \).

Once an MDP \( \bar{M} \) has been specified, any standard planning algorithm can be used to obtain the value function \( \bar{V}^* \). In the next section, we shall show that using a particular implementation of VI for planning has the advantage of being differentiable, and simple to implement within a NN framework. In this section however, we focus on how to use the planning result \( \bar{V}^* \) within the NN policy \( \pi \). Our approach is based on two important observations. The first is that the vector of values \( \bar{V}^*(s) \) \( \bar{S} \) encodes all the information about the optimal plan in \( \bar{M} \). Thus, adding the vector \( \bar{V}^* \) as additional features to the policy \( \pi \) is sufficient for extracting information about the optimal plan in \( \bar{M} \).

However, an additional property of \( \bar{V}^* \) is that the optimal decision \( \pi^*(\bar{s}) \) at a state \( \bar{s} \) can depend only on a subset of the values of \( \bar{V}^* \), since \( \pi^*(\bar{s}) = \arg \max_a \bar{R}(\bar{s}, a) + \gamma \sum_{s'} \bar{P}(\bar{s}' | \bar{s}, a) \bar{V}^*(\bar{s}') \).

Therefore, if the MDP has a local connectivity structure, such as in the grid-world example above, the states for which \( \bar{P}(\bar{s}' | \bar{s}, a) > 0 \) is a small subset of \( \bar{S} \).

In NN terminology, this is a form of attention [31], in the sense that for a given label prediction (action), only a subset of the input features (value function) is relevant. Attention is known to improve learning performance by reducing the effective number of network parameters during learning.
Therefore, the second element in our network is an attention module that outputs a vector of (attention modulated) values $\psi(s)$. Finally, the vector $\psi(s)$ is added as additional features to a reactive policy $\pi_\theta(a|\phi(s), \psi(s))$. The full network architecture is depicted in Figure 2 (left).

Returning to our grid-world example, at a particular state $s$, the reactive policy only needs to query the values of the states neighboring $s$ in order to select the correct action. Thus, the attention module in this case could return a $\psi(s)$ vector with a subset of $V^*$ for these neighboring states.

Figure 2: Planning-based NN models. Left: a general policy representation that adds value function features from a planner to a reactive policy. Right: VI module – a CNN representation of VI algorithm.

Let $\theta$ denote all the parameters of the policy, namely, the parameters of $f_R$, $f_P$, and $\pi_\theta$, and note that $\psi(s)$ is in fact a function of $\phi(s)$. Therefore, the policy can be written in the form $\pi_\theta(a|\phi(s))$, similarly to the standard policy form (cf. Section 2). If we could back-propagate through this function, then potentially we could train the policy using standard RL and IL algorithms, just like any other standard policy representation. While it is easy to design functions $f_R$ and $f_P$ that are differentiable (and we provide several examples in our experiments), back-propagating the gradient through the planning algorithm is not trivial. In the following, we propose a novel interpretation of an approximate VI algorithm as a particular form of a CNN. This allows us to conveniently treat the planning module as just another NN, and by back-propagating through it, we can train the whole policy end-to-end.

### 3.1 The VI Module

We now introduce the VI module – a NN that encodes a differentiable planning computation. Our starting point is the VI algorithm (1). Our main observation is that each iteration of VI may be seen as passing the previous value function $V^n$ and reward function $R$ through a convolution layer and max-pooling layer. In this analogy, each channel in the convolution layer corresponds to the $Q$-function for a specific action, and convolution kernel weights correspond to the discounted transition probabilities. Thus by recurrently applying a convolution layer $K$ times, $K$ iterations of VI are effectively performed.

Following this idea, we propose the VI network module, as depicted in Figure 2B. The inputs to the VI module are a ‘reward image’ $\tilde{R}$ of dimensions $l, m, n$, where here, for the purpose of clarity, we follow the CNN formulation and explicitly assume that the state space $S$ maps to a 2-dimensional grid. However, our approach can be extended to general discrete state spaces, for example, a graph, as we report in the WikiNav experiment in Section 4.4. The reward is fed into a convolutional layer $Q$ with $A$ channels and a linear activation function, $Q_{a,i',j'} = \sum_{l,i,j} W_{a,l,i,j} \tilde{R}_{l,i'-i,j'-j}$. Each channel $a$ in this layer corresponds to $Q(s, a)$ for a particular action $a$. This layer is then max-pooled along the actions channel to produce the next-iteration value function layer $V$, $V_{i,j} = \max_a Q(a, i, j)$. The next-iteration value function layer $V$ is then stacked with the reward $R$, and fed back into the convolutional layer and max-pooling layer $K$ times, to perform $K$ iterations of value iteration.

The VI module is simply a NN architecture that has the capability of performing an approximate VI computation. Nevertheless, representing VI in this form makes learning the MDP parameters and reward function natural – by backpropagating through the network, similarly to a standard CNN. VI modules can also be composed hierarchically, by treating the value of one VI module as additional input to another VI module. We further report on this idea in the supplementary material.

### 3.2 Value Iteration Networks

We now have all the ingredients for a differentiable planning-based policy, which we term a value iteration network (VIN). The VIN is based on the general planning-based policy defined above, with
We design a VIN for this task following the guidelines described above, where the planning MDP $\bar{M}$.

Once a VIN design is chosen, implementing the VIN is straightforward, as it is simply a form of a CNN. The networks in our experiments all required only several lines of Theano [1] code. In the next section, we evaluate VIN policies on various domains, showing that by learning to plan, they achieve a better generalization capability.

4 Experiments

In this section we evaluate VINs as policy representations on various domains. Additional experiments investigating RL and hierarchical VINs, as well as technical implementation details are discussed in the supplementary material. Source code is available at https://github.com/avivt/VIN. Our goal in these experiments is to investigate the following questions:

1. Can VINs effectively learn a planning computation using standard RL and IL algorithms?
2. Does the planning computation learned by VINs make them better than reactive policies at generalizing to new domains?

An additional goal is to point out several ideas for designing VINs for various tasks. While this is not an exhaustive list that fits all domains, we hope that it will motivate creative designs in future work.

4.1 Grid-World Domain

Our first experiment domain is a synthetic grid-world with randomly placed obstacles, in which the observation includes the position of the agent, and also an image of the map of obstacles and goal position. Figure 3 shows two random instances of such a grid-world of size $16 \times 16$. We conjecture that by learning the optimal policy for several instances of this domain, a VIN policy would learn the planning computation required to solve a new, unseen, task.

In such a simple domain, an optimal policy can easily be calculated using exact VI. Note, however, that here we are interested in evaluating whether a NN policy, trained using RL or IL, can learn to plan. In the following results, policies were trained using IL, by standard supervised learning from demonstrations of the optimal policy. In the supplementary material, we report additional RL experiments that show similar findings.

We design a VIN for this task following the guidelines described above, where the planning MDP $\bar{M}$ is a grid-world, similar to the true MDP. The reward mapping $f_R$ is a CNN mapping the image input to a reward map in the grid-world. Thus, $f_R$ should potentially learn to discriminate between obstacles, non-obstacles and the goal, and assign a suitable reward to each. The transitions $\bar{P}$ were defined as $3 \times 3$ convolution kernels in the VI block, exploiting the fact that transitions in the grid-world are local. The recurrence $K$ was chosen in proportion to the grid-world size, to ensure that information can flow from the goal state to any other state. For the attention module, we chose a trivial approach that selects the $Q$ values in the VI block for the current state, i.e., $\psi(s) = Q(s, \cdot)$. The final reactive policy is a fully connected network that maps $\psi(s)$ to a probability over actions.

We compare VINs to the following NN reactive policies:

CNN network: We devised a CNN-based reactive policy inspired by the recent impressive results of DQN [21], with 5 convolution layers, and a fully connected output. While the network in [21] was trained to predict Q values, our network outputs a probability over actions. These terms are related, since $\pi^*(s) = \arg\max_{a} Q(s, a)$. Fully Convolutional Network (FCN): The problem setting for this domain is similar to semantic segmentation [19], in which each pixel in the image is assigned a semantic label (the action in our case). We therefore devised an FCN inspired by a state-of-the-art semantic segmentation algorithm [19], with 3 convolution layers, where the first layer has a filter that spans the whole image, to properly convey information from the goal to every other state.

In Table 1 we present the average $0-1$ prediction loss of each model, evaluated on a held-out test-set of maps with random obstacles, goals, and initial states, for different problem sizes. In addition, for

---

2VINs are fundamentally different than inverse RL methods [22], where transitions are required to be known.

3Note that the transitions defined this way do not depend on the state $s$. Interestingly, we shall see that the network learned to plan successful trajectories nevertheless, by appropriately shaping the reward.
Table 1: Performance on grid-world domain. Top: comparison with reactive policies. For all domain sizes, VIN networks significantly outperform standard reactive networks. Note that the performance gap increases dramatically with problem size.

<table>
<thead>
<tr>
<th>Domain</th>
<th>VIN</th>
<th>CNN</th>
<th>FCN</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 × 8</td>
<td>0.004</td>
<td>99.6%</td>
<td>0.001</td>
</tr>
<tr>
<td>16 × 16</td>
<td>0.05</td>
<td>99.3%</td>
<td>0.089</td>
</tr>
<tr>
<td>28 × 28</td>
<td>0.11</td>
<td>97%</td>
<td>0.086</td>
</tr>
</tbody>
</table>

4.2 Mars Rover Navigation

In this experiment we show that VINs can learn to plan from natural image input. We demonstrate this on path-planning from overhead terrain images of a Mars landscape.

Each domain is represented by a 128 × 128 image patch, on which we defined a 16 × 16 grid-world, where each state was considered an obstacle if the terrain in its corresponding 8 × 8 image patch contained an elevation angle of 10 degrees or more, evaluated using an external elevation data base. An example of the domain and terrain image is depicted in Figure 3. The MDP for shortest-path planning in this case is similar to the grid-world domain of Section 4.1, and the VIN design was similar, only with a deeper CNN in the reward mapping \( f_R \) for processing the image.

The policy was trained to predict the shortest-path directly from the terrain image. We emphasize that the elevation data is not part of the input, and must be inferred (if needed) from the terrain image.
After training, VIN achieved a success rate of 84.8%. To put this rate in context, we compare with the best performance achievable without access to the elevation data, which is 90.3%. To make this comparison, we trained a CNN to classify whether an 8 × 8 patch is an obstacle or not. This classifier was trained using the same image data as the VIN network, but its labels were the true obstacle classifications from the elevation map (we reiterate that the VIN did not have access to these ground-truth obstacle labels during training or testing). The success rate of planner that uses the obstacle map generated by this classifier from the raw image is 90.3%, showing that obstacle identification from the raw image is indeed challenging. Thus, the success rate of the VIN, which was trained without any obstacle labels, and had to ‘figure out’ the planning process is quite remarkable.

### 4.3 Continuous Control

We now consider a 2D path planning domain with continuous states and continuous actions, which cannot be solved using VI, and therefore a VIN cannot be naively applied. Instead, we will construct the VIN to perform ‘high-level’ planning on a discrete, coarse, grid-world representation of the continuous domain. We shall show that a VIN can learn to plan such a ‘high-level’ plan, and also exploit that plan within its ‘low-level’ continuous control policy. Moreover, the VIN policy results in better generalization than a reactive policy.

Consider the domain in Figure 4. A red-colored particle needs to be navigated to a green goal using horizontal and vertical forces. Gray-colored obstacles are randomly positioned in the domain, and apply an elastic force and friction when contacted. This domain presents a non-trivial control problem, as the agent needs to both plan a feasible trajectory between the obstacles (or use them to bounce off), but also control the particle (which has mass and inertia) to follow it. The state observation consists of the particle’s continuous position and velocity, and a static 16 × 16 downscaled image of the obstacles and goal position in the domain. In principle, such an observation is sufficient to devise a ‘rough plan’ for the particle to follow.

As in our previous experiments, we investigate whether a policy trained on several instances of this domain with different start state, goal, and obstacle positions, would generalize to an unseen domain. For training we chose the guided policy search (GPS) algorithm with unknown dynamics [17], which is suitable for learning policies for continuous dynamics with contacts, and we used the publicly available GPS code [8], and Mujoco [29] for physical simulation. We generated 200 random training instances, and evaluate our performance on 40 different test instances from the same distribution.

Our VIN design is similar to the grid-world cases, with some important modifications: the attention module selects a 5 × 5 patch of the value $\bar{V}$, centered around the current (discretized) position in the map. The final reactive policy is a 3-layer fully connected network, with a 2-dimensional continuous output for the controls. In addition, due to the limited number of training domains, we pre-trained the VIN with transition weights that correspond to discounted grid-world transitions. This is a reasonable prior for the weights in a 2-d task, and we emphasize that even with this initialization, the initial value function is meaningless, since the reward map $f_R$ is not yet learned. We compare with a CNN-based reactive policy inspired by the state-of-the-art results in [21, 20], with 2 CNN layers for image processing, followed by a 3-layer fully connected network similar to the VIN reactive policy.

Figure 4 shows the performance of the trained policies, measured as the final distance to the target. The VIN clearly outperforms the CNN on test domains. We also plot several trajectories of both policies on test domains, showing that VIN learned a more sensible generalization of the task.

### 4.4 WebNav Challenge

In the previous experiments, the planning aspect of the task corresponded to 2D navigation. We now consider a more general domain: WebNav [23] – a language based search task on a graph.

In WebNav [23], the agent needs to navigate the links of a website towards a goal web-page, specified by a short 4-sentence query. At each state $s$ (web-page), the agent can observe average word-embedding features of the state $\phi(s)$ and possible next states $\phi(s')$ (linked pages), and the features of
We remark that there is still room for further improvements of the WebNav results, e.g., by better planning, based on a VIN, can lead to better performance in this task, when starting from the root node, achieving 1030 successful runs vs. 1025 for the baseline. However, after training, the VIN policy performed mildly better than the baseline on 2000 held-out test queries. These results confirm that indeed, when navigating a tree of categories from the root up, the features at each state contain meaningful information about the path to the goal. Indeed, this property naturally holds for an encyclopedic website that is structured as a tree of categories, sub-categories, sub-sub-categories, etc.

We sought to explore whether planning, based on a VIN, can lead to better performance in this task, with the intuition that a plan on a simplified model of the website can help guide the reactive policy in difficult queries. Therefore, we designed a VIN that plans on a small subset of the graph that contains only the 1st and 2nd level categories (< 3% of the graph), and their word-embedding features.

Designing this VIN requires a different approach from the grid-world VINs described earlier, where the most challenging aspect is to define a meaningful mapping between nodes in the true graph and nodes in the smaller VIN graph. For the reward mapping \( f_R \), we chose a weighted similarity measure between the query features \( \phi(q) \), and the features of nodes in the small graph \( \phi(s) \). Thus, intuitively, nodes that are similar to the query should have high reward. The transitions were fixed based on the graph connectivity of the smaller VIN graph, which is known, though different from the true graph. The attention module was also based on a weighted similarity measure between the features of the possible next states \( \phi(s') \) and the features of each node in the simplified graph \( \phi(\tilde{s}) \). The reactive policy part of the VIN was similar to the policy of [23] described above. Note that by training such a VIN end-to-end, we are effectively learning how to exploit the small graph for doing better planning on the true, large graph.

Both the VIN policy and the baseline reactive policy were trained by supervised learning, on random trajectories that start from the root node of the graph. Similarly to [23], a policy is said to succeed a query if all the correct predictions along the path are within its top-4 predictions.

After training, the VIN policy performed mildly better than the baseline on 2000 held-out test queries when starting from the root node, achieving 1030 successful runs vs. 1025 for the baseline. However, when we tested the policies on a harder task of starting from a random position in the graph, VINs significantly outperformed the baseline, achieving 346 successful runs vs. 304 for the baseline, out of 4000 test queries. These results confirm that indeed, when navigating a tree of categories from the root up, the features at each state contain meaningful information about the path to the goal, making a reactive policy sufficient. However, when starting the navigation from a different state, a reactive policy may fail to understand that it needs to first go back to the root and switch to a different branch in the tree. Our results indicate such a strategy can be better represented by a VIN.

We remark that there is still room for further improvements of the WebNav results, e.g., by better models for reward and attention functions, and better word-embedding representations of text.

5 Conclusion and Outlook

The introduction of powerful and scalable RL methods has opened up a range of new problems for deep learning. However, few recent works investigate policy architectures that are specifically tailored for planning under uncertainty, and current RL theory and benchmarks rarely investigate the generalization properties of a trained policy [27, 21, 6]. This work takes a step in this direction, by exploring better generalizing policy representations.

Our VIN policies learn an approximate planning computation relevant for solving the task, and we have shown that such a computation leads to better generalization in a diverse set of tasks, ranging from simple gridworlds that are amenable to value iteration, to continuous control, and even to navigation of Wikipedia links. In future work we intend to learn different planning computations, based on simulation [11], or optimal linear control [30], and combine them with reactive policies, to potentially develop RL solutions for task and motion planning [14].

Acknowledgments

This research was funded in part by Siemens, by ONR through a PECASE award, by the Army Research Office through the MAST program, and by an NSF CAREER grant. A. T. was partially funded by the Viterbi Scholarship, Technion. Y. W. was partially funded by a DARPA PPAML program, contract FA8750-14-C-0011.
References


