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 [3, 15, 6, 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) [9]. 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* [2]. However, most recent deep RL works [21, 18, 9] 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 task[1].

---

[1] In 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 [1, 2] 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.

Figure 1: Two instances of a grid-world domain. Task is to move to the goal between the obstacles.

Connections between planning algorithms and recurrent NNs were previously explored by Ilin et al. [12]. Our work builds on related ideas, but results in a more broadly applicable policy representation. Our approach is different from model-based RL [25, 4], 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, 11].

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) [1, 2]. An MDP M consists of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, 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 π is the expected discounted sum of rewards when starting from that state and executing policy π, $V^\pi(s) \doteq \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s \right]$, where $\gamma \in (0, 1)$ is a discount factor, and $\mathbb{E}^\pi$ denotes an expectation over trajectories of states and actions $(s_0, a_0, s_1, a_1 \dots)$, in which actions are selected according to π, and states evolve according to the transition kernel $P(s'|s, a)$. The optimal value function $V^*(s) \doteq \max_\pi V^\pi(s)$ is the maximal long-term return possible from a state. A policy π^* is said to be optimal if $V^{\pi^*}(s) = V^*(s) \ \forall s$. A popular algorithm for calculating V^* and π^* is value iteration (VI):

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

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 [8, 16, 3, 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 $W^1, \dots, W^{l'}$, $h_{l', i', j'} = \sigma \left(\sum_{l,i,j} W^{l'}_{l,i,j} X_{l,i'-i,j'-j} \right)$, where σ 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 $N(i, j)$, $h^{maxpool}_{l,i,j} = \max_{i',j' \in N(i,j)} h_{l,i',j'}$. Typically, the neighbors $N(i, j)$ are chosen as a $k \times k$ image

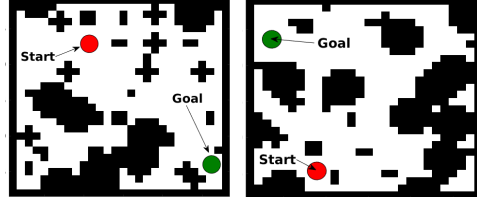patch around pixel i, j. After max-pooling, the image is down-sampled by a constant factor d, commonly 2 or 4, resulting in an output signal with l' channels, m/d horizontal pixels, and n/d vertical pixels. CNNs are typically trained using stochastic gradient descent (SGD), with backpropagation for computing gradients.

Reinforcement Learning and Imitation Learning: In MDPs where the state space is very large or continuous, or when the MDP transitions or rewards are not known in advance, planning algorithms cannot be applied. In these cases, a policy can be *learned* from either expert supervision – IL, or by trial and error – RL. While the learning algorithms in both cases are different, the policy representations – which are the focus of this work – are similar. Additionally, most state-of-the-art algorithms such as [24, 21, 26, 18] are agnostic to the policy representation, and only require it to be differentiable, for performing gradient descent on some algorithm-specific loss function. Therefore, in this paper we do not commit to a specific learning algorithm, and only consider the policy.

Let $\phi(s)$ denote an observation for state s. The policy is specified as a parametrized function $\pi_\theta(a|\phi(s))$ mapping observations to a probability over actions, where θ are the policy parameters. For example, the policy could be represented as a neural network, with θ denoting the network weights. The goal is to tune the parameters such that the policy behaves well in the sense that $\pi_\theta(a|\phi(s)) \approx \pi^*(a|\phi(s))$, where π^* is the optimal policy for the MDP, as defined in Section 2.

In IL, a dataset of N state observations and corresponding optimal actions $\left\{\phi(s^i), a^i \sim \pi^*(\phi(s^i))\right\}_{i=1,\ldots,N}$ is generated by an expert. Learning a policy then becomes an instance of supervised learning [24, 9]. 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 π. 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* $\bar{M}$, and to add the solution of $\bar{M}$ as an element in the policy π. We hypothesize that this will lead to a policy that automatically learns a useful $\bar{M}$ to plan on. We denote by $\bar{s} \in \bar{S}$, $\bar{a} \in \bar{A}$, $\bar{R}(\bar{s}, \bar{a})$, and $\bar{P}(\bar{s}'|\bar{s}, \bar{a})$ the states, actions, rewards, and transitions in $\bar{M}$. To facilitate a connection between M and $\bar{M}$, we let $\bar{R}$ and $\bar{P}$ depend on the observation in M, namely, $\bar{R} = f_R(\phi(s))$ and $\bar{P} = f_P(\phi(s))$, 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 $\bar{M}$ 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 π. Our approach is based on two important observations. The first is that the vector of values $\bar{V}^*(s) \;\; \forall s$ encodes all the information about the optimal plan in $\bar{M}$. Thus, adding the vector $\bar{V}^*$ as additional features to the policy π is sufficient for extracting information about the optimal plan in $\bar{M}$.

However, an additional property of $\bar{V}^*$ is that the optimal decision $\bar{\pi}^*(\bar{s})$ at a state $\bar{s}$ can depend only on a subset of the values of $\bar{V}^*$, since $\bar{\pi}^*(\bar{s}) = \arg\max_{\bar{a}} \bar{R}(\bar{s}, \bar{a}) + \gamma \sum_{\bar{s}'} \bar{P}(\bar{s}'|\bar{s}, \bar{a}) \bar{V}^*(\bar{s}')$. Therefore, if the MDP has a local connectivity structure, such as in the grid-world example above, the states for which $\bar{P}(\bar{s}'|\bar{s}, \bar{a}) > 0$ is a small subset of $\bar{S}$.

In NN terminology, this is a form of *attention* [32], 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 $\pi_{\text{re}}(a|\phi(s), \psi(s))$. 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 $\bar{V}^*$ for these neighboring states.
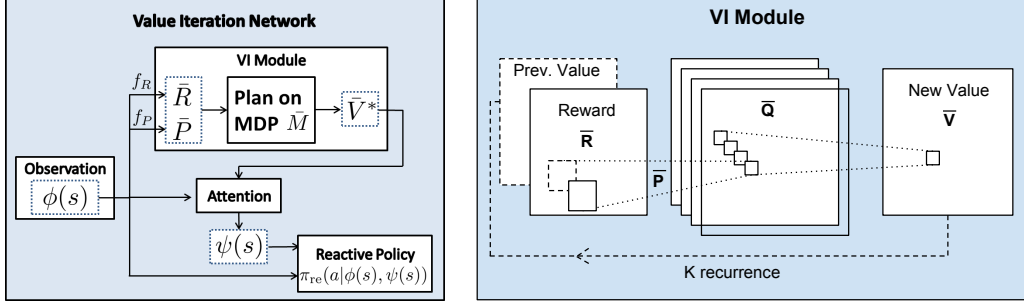

Figure 2: Planning-based NN models. Left: a general policy representation that adds value function features from a planner to a reactive policy. Right: VI module – a CNN representation of VI algorithm.

Let θ denote all the parameters of the policy, namely, the parameters of f_R, f_P, and π_{re}, 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 is a 'reward image' $\bar{R}$ of dimensions l, m, n, where here, for the purpose of clarity, we follow the CNN formulation and explicitly assume that the state space $\bar{S}$ 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 $\bar{Q}$ with $\bar{A}$ channels and a linear activation function, $\bar{Q}_{\bar{a},i',j'} = \sum_{l,i,j} W^{\bar{a}}_{l,i,j} \bar{R}_{l,i'-i,j'-j}$. Each channel in this layer corresponds to $\bar{Q}(\bar{s}, \bar{a})$ for a particular action $\bar{a}$. This layer is then max-pooled along the actions channel to produce the next-iteration value function layer $\bar{V}$, $\bar{V}_{i,j} = \max_{\bar{a}} \bar{Q}(\bar{a}, i, j)$. The next-iteration value function layer $\bar{V}$ is then stacked with the reward $\bar{R}$, 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 the VI module as the planning algorithm. In order to implement a VIN, one has to specify the state

and action spaces for the planning module $\bar{S}$ and $\bar{A}$, the reward and transition functions f_R and f_P, and the attention function; we refer to this as the *VIN design*. For some tasks, as we show in our experiments, it is relatively straightforward to select a suitable design, while other tasks may require more thought. However, we emphasize an important point: the reward, transitions, and attention can be defined by parametric functions, and trained with the whole policy[2]. Thus, a rough design can be specified, and then fine-tuned by end-to-end training.

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 [28] 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 16×16. 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 3×3 convolution kernels in the VI block, exploiting the fact that transitions in the grid-world are local[3]. 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 $\bar{Q}$ values in the VI block for the current state, i.e., $\psi(s) = \bar{Q}(s, \cdot)$. 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 each map, a full trajectory from the initial state was predicted, by iteratively rolling-out the next-states

---

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

[3]Note that the transitions defined this way do not depend on the state $\bar{s}$. Interestingly, we shall see that the network learned to plan successful trajectories nevertheless, by appropriately shaping the reward.
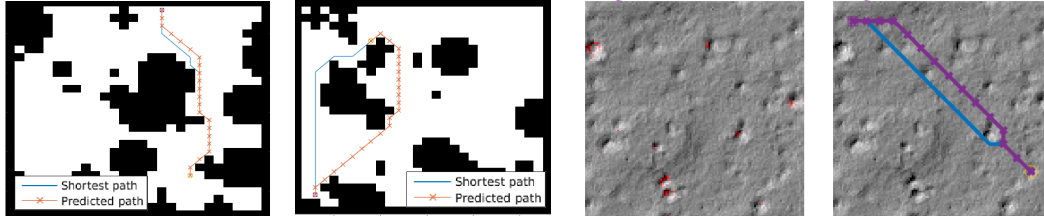
Figure 3: Grid-world domains (best viewed in color). A,B: Two random instances of the 28×28 synthetic gridworld, with the VIN-predicted trajectories and ground-truth shortest paths between random start and goal positions. C: An image of the Mars domain, with points of elevation sharper than $10°$ colored in red. These points were calculated from a matching image of elevation data (not shown), and were not available to the learning algorithm. Note the difficulty of distinguishing between obstacles and non-obstacles. D: The VIN-predicted (purple line with cross markers), and the shortest-path ground truth (blue line) trajectories between between random start and goal positions.

Domain	VIN			CNN			FCN		
	Prediction loss	Success rate	Traj. diff.	Pred. loss	Succ. rate	Traj. diff.	Pred. loss	Succ. rate	Traj. diff.
8×8	0.004	**99.6%**	0.001	0.02	97.9%	0.006	0.01	97.3%	0.004
16×16	0.05	**99.3%**	0.089	0.10	87.6%	0.06	0.07	88.3%	0.05
28×28	0.11	**97%**	0.086	0.13	74.2%	0.078	0.09	76.6%	0.08

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.

predicted by the network. A trajectory was said to succeed if it reached the goal without hitting obstacles. For each trajectory that succeeded, we also measured its difference in length from the optimal trajectory. The average difference and the average success rate are reported in Table 1.

Clearly, VIN policies generalize to domains outside the training set. A visualization of the reward mapping f_R (see supplementary material) shows that it is negative at obstacles, positive at the goal, and a small negative constant otherwise. The resulting value function has a gradient pointing towards a direction to the goal around obstacles, thus a useful planning computation was learned. VINs also significantly outperform the reactive networks, and the performance gap increases dramatically with the problem size. Importantly, note that the prediction loss for the reactive policies is comparable to the VINs, although their success rate is significantly worse. This shows that this is not a standard case of overfitting/underfitting of the reactive policies. Rather, VIN policies, by their VI structure, focus prediction errors on less important parts of the trajectory, while reactive policies do not make this distinction, and learn the easily predictable parts of the trajectory yet fail on the complete task.

The VINs have an effective depth of K, which is larger than the depth of the reactive policies. One may wonder, whether any deep enough network would learn to plan. In principle, a CNN or FCN of depth K has the potential to perform the same computation as a VIN. However, it has much more parameters, requiring much more training data. We evaluate this by untying the weights in the K recurrent layers in the VIN. Our results, reported in the supplementary material, show that untying the weights degrades performance, with a stronger effect for smaller sizes of training data.

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.

Network	Train Error	Test Error
VIN	0.30	0.35
CNN	0.39	0.59

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,
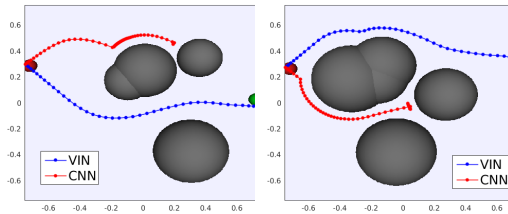
Figure 4: Continuous control domain. Top: average distance to goal on training and test domains for VIN and CNN policies. Bottom: trajectories predicted by VIN and CNN on test domains.

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 [7], and Mujoco [30] 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 the query $\phi(q)$, and based on that has to select which link to follow. In [23], the search was performed

on the Wikipedia website. Here, we report experiments on the 'Wikipedia for Schools' website, a simplified Wikipedia designed for children, with over 6000 pages and at most 292 links per page.

In [23], a NN-based policy was proposed, which first learns a NN mapping from $(\phi(s), \phi(q))$ to a hidden state vector h. The action is then selected according to $\pi(s'|\phi(s), \phi(q)) \propto \exp\left(h^\top \phi(s')\right)$. In essence, this policy is reactive, and relies on the word embedding features at each state to 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 $\phi(\bar{s})$. 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 $\phi(\bar{s})$. 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, 5]. 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 [10], or optimal linear control [31], and combine them with reactive policies, to potentially develop RL solutions for task and motion planning [14].

References

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] D. Bertsekas. *Dynamic Programming and Optimal Control, Vol II*. Athena Scientific, 4th edition, 2012.

[3] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition*, pages 3642–3649, 2012.

[4] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[5] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*, 2016.

[6] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, 2013.

[7] C. Finn, M. Zhang, J. Fu, X. Tan, Z. McCarthy, E. Scharff, and S. Levine. Guided policy search code implementation, 2016. Software available from rll.berkeley.edu/gps.

[8] K. Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position- neocognitron. *Transactions of the IECE*, J62-A(10):658–665, 1979.

[9] A. Giusti et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 2016.

[10] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *NIPS*, 2014.

[11] X. Guo, S. Singh, R. Lewis, and H. Lee. Deep learning for reward design to improve monte carlo tree search in atari games. *arXiv:1604.07095*, 2016.

[12] R. Ilin, R. Kozma, and P. J. Werbos. Efficient learning in cellular simultaneous recurrent neural networks-the case of maze navigation problem. In *ADPRL*, 2007.

[13] J. Joseph, A. Geramifard, J. W. Roberts, J. P. How, and N. Roy. Reinforcement learning with misspecified model classes. In *ICRA*, 2013.

[14] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1470–1477, 2011.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, 2014.

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17, 2016.

[19] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[22] G. Neu and C. Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *UAI*, 2007.

[23] R. Nogueira and K. Cho. Webnav: A new large-scale task for natural language based sequential decision making. *arXiv preprint arXiv:1602.02261*, 2016.

[24] S. Ross, G. Gordon, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[25] J. Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *International Joint Conference on Neural Networks*. IEEE, 1990.

[26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *ICML*, 2015.

[27] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[28] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[29] T. Tieleman and G. Hinton. Lecture 6.5. COURSERA: Neural Networks for Machine Learning, 2012.

[30] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[31] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, 2015.

[32] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

A   Visualization of Learned Reward and Value

In Figure 5 we plot the learned reward and value function for the gridworld task. The learned reward is very negative at obstacles, very positive at goal, and a slightly negative constant otherwise. The resulting value function has a peak at the goal, and a gradient pointing towards a direction to the goal around obstacles. This plot clearly shows that the VI block learned a useful planning computation.
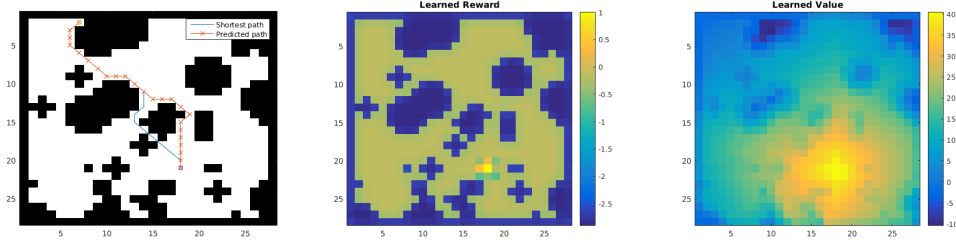

Figure 5: Visualization of learned reward and value function. Left: a sample domain. Center: learned reward f_R for this domain. Right: resulting value function (in VI block) for this domain.

B   Weight Sharing

The VINs have an effective depth of K, which is larger than the depth of the reactive policies. One may wonder, whether any deep enough network would learn to plan. In principle, a CNN or FCN of depth K has the potential to perform the same computation as a VIN. However, it has much more parameters, requiring much more training data. We evaluate this by untying the weights in the K recurrent layers in the VIN. Our results, in Table 2 show that untying the weights degrades performance, with a stronger effect for smaller sizes of training data.

Training data	VIN			VIN Untied Weights		
	Pred. loss	Succ. rate	Traj. diff.	Pred. loss	Succ. rate	Traj. diff.
20%	0.06	98.2%	0.106	0.09	91.9%	0.094
50%	0.05	99.4%	0.018	0.07	95.2%	0.078
100%	0.05	99.3%	0.089	0.05	95.6%	0.068

Table 2: Performance on 16×16 grid-world domain. Evaluation of the effect of VI module shared weights relative to data size.

C   Gridworld with Reinforcement Learning

We demonstrate that the value iteration network can be trained using reinforcement learning methods and achieves favorable generalization properties as compared to standard convolutional neural networks (CNNs).

The overall setup of the experiment is as follows: we train policies parameterized by VINs and policies parameterized by convolutional networks on the same set of randomly generated gridworld maps in the same way (described below) and then test their performance on a held-out set of test maps, which was generated in the same way as the set of training maps but is disjoint from the training set.

The MDP is what one would expect of a gridworld environment – the states are the positions on the map; the actions are movements up, down, left, and right; the rewards are $+1$ for reaching the goal, -1 for falling into a hole, and -0.01 otherwise (to encourage the policy to find the shortest path); the transitions are deterministic.

Structure of the networks. The VINs used are similar to those described in the main body of the paper. After K value-iteration recurrences, we have approximate Q values for every state and action in the map. The attention selects only those for the current state, and these are converted to a

Network	8×8	16×16
VIN	90.9%	82.5%
CNN	86.9%	33.1%

Table 3: RL Results – performance on test maps.

probability distribution over actions using the softmax function. We use $K = 10$ for the 8×8 maps and $K = 20$ for the 16×16 maps.

The convolutional networks' structure was adapted to accommodate the size of the maps. For the 8×8 maps, we use 50 filters in the first layer and then 100 filters in the second layer, all of size 3×3. Each of these layers is followed by a 2×2 max-pool. At the end we have a fully connected hidden layer with 100 hidden units, followed by a fully-connected layer to the (4) outputs, which are converted to probabilities using the softmax function.

The network for the 16×16 maps is similar but uses three convolutional layers (with 50, 100, and 100 filters respectively), the first two of which are 2×2 max-pooled, followed by two fully-connected hidden layers (200 and 100 hidden units respectively) before connecting to the outputs and performing softmax.

Training with a curriculum. To ensure that the policies are not simply memorizing specific maps, we randomly select a map before each episode. But some maps are far more difficult than others, and the agent learns best when it stands a reasonable chance of reaching this goal. Thus we found it beneficial to begin training on the easiest maps and then gradually progress to more difficult maps. This is the idea of *curriculum training*.

We consider curriculum training as a way to address the exploration problem. If a completely untrained agent is dropped into a very challenging map, it moves randomly and stands approximately zero chance of reaching the goal (and thus learning a useful reward). But even a random policy can consistently reach goals nearby and learn something useful in the process, e.g. to move toward the goal. Once the policy knows how to solve tasks of difficulty n, it can more easily learn to solve tasks of difficulty $n + 1$, as compared to a completely untrained policy. This strategy is well-aligned with how formal education is structured; you can't effectively learn calculus without knowing basic algebra.

Not all environments have an obvious difficulty metric, but fortunately the gridworld task does. We define the difficulty of a map as the length of the shortest path from the start state to the goal state. It is natural to start with difficulty 1 (the start state and goal state are adjacent) and ramp up the difficulty by one level once a certain threshold of "success" is reached. In our experiments we use the average discounted return to assess progress and increase the difficulty level from n to $n + 1$ when the average discounted return for an iteration exceeds $1 - \frac{n}{35}$. This rule was chosen empirically and takes into account the fact that higher difficulty levels are more difficult to learn.

All networks were trained using the trust region policy optimization (TRPO) [26] algorithm, using publicly available code in the RLLab benchmark [5].

Testing. When testing, we ignore the exact rewards and measure simply whether or not the agent reaches the goal. For each map in the test set, we run an episode, noting if the policy succeeds in reaching the goal. The proportion of successful trials out of all the trials is reported for each network. (See Table 3.)

On the 8×8 maps, we used the same number of training iterations on both types of networks to make the comparison as fair as possible. On the 16×16 maps, it became clear that the convolutional network was struggling, so we allowed it twice as many training iterations as the VIN, yet it still failed to achieve even a remotely similar level of performance on the test maps. (See left image of Figure 6.) We posit that this is because the VIN learns to plan, while the CNN simply follows a reactive policy. Though the CNN policy performs reasonably well on the smaller domains, it does not scale to larger domains, while the VIN does. (See right image of Figure 6.)

D Technical Details for Experiments

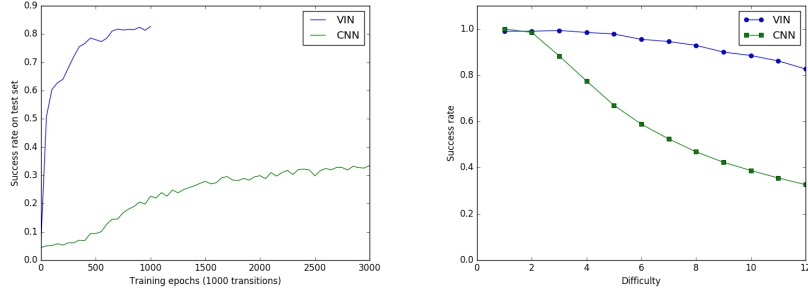We report the full technical details used for training our networks.

Figure 6: RL results – performance of VIN and CNN on 16×16 test maps. Left: Performance on all maps as a function of amount of training. Right: Success rate on test maps of increasing difficulty.

D.1 Grid-world Domain

Our training set consists of $N_i = 5000$ random grid-world instances, with $N_t = 7$ shortest-path trajectories (calculated using an optimal planning algorithm) from a random start-state to a random goal-state for each instance; a total of $N_i \times N_t$ trajectories. For each state $s = (i, j)$ in each trajectory, we produce a $(2 \times m \times n)$-sized observation image s_{image}. The first channel of s_{image} encodes the obstacle presence (1 for obstacle, 0 otherwise), while the second channel encodes the goal position (1 at the goal, 0 otherwise). The full observation vector is $\phi(s) = [s, s_{\text{image}}]$. In addition, for each state we produce a label a that encodes the action (one of 8 directions) that an optimal shortest-path policy would take in that state.

We design a VIN for this task as follows. The state space $\bar{S}$ was chosen to be a $m \times n$ grid-world, similar to the true state space S.[4] The reward $\bar{R}$ in this space can be represented by an $m \times n$ map, and we chose the reward mapping f_R to be a CNN with s_{image} as its input, one layer with 150 kernels of size 3×3, and a second layer with one 3×3 filter to output $\bar{R}$. Thus, f_R maps the image of obstacles and goal to a 'reward image'. The transitions $\bar{P}$ were defined as 3×3 convolution kernels in the VI block, and exploit the fact that transitions in the grid-world are local. Note that the transitions defined this way do not depend on the state $\bar{s}$. Interestingly, we shall see that the network learned rewards and transitions that nevertheless enable it to successfully plan in this task. For the attention module, since there is a one-to-one mapping between the agent position in S and in $\bar{S}$, we chose a trivial approach that selects the $\bar{Q}$ values in the VI block for the state in the real MDP s, i.e., $\psi(s) = \bar{Q}(s, \cdot)$. The final reactive policy is a fully connected softmax output layer with weights W, $\pi_{\text{re}}(\cdot | \psi(s)) \propto \exp\left(W^\top \psi(s)\right)$.

We trained several neural-network policies based on a multi-class logistic regression loss function using stochastic gradient descent, with an RMSProp step size [29], implemented in the Theano [28] library.

We compare the policies:

VIN network We used the VIN model of Section 3 as described above, with 10 channels for the q layer in the VI block. The recurrence K was set relative to the problem size: $K = 10$ for 8×8 domains, $K = 20$ for 16×16 domains, and $K = 36$ for 28×28 domains. The guideline for choosing these values was to keep the network small while guaranteeing that goal information can flow to every state in the map.

CNN network: We devised a CNN-based reactive policy inspired by the recent impressive results of DQN [21], with 5 convolution layers with $[50, 50, 100, 100, 100]$ kernels of size 3×3, and 2×2 max-pooling after the first and third layers. The final layer is fully connected, and maps to a softmax over actions. To represent the current state, we added to s_{image} a channel that encodes the current position (1 at the current state, 0 otherwise).

---

[4]For a particular configuration of obstacles, the true grid-world domain can be captured by a $m \times n$ state space with the obstacles encoded in the MDP transitions, as in our notation. For a general obstacle configuration, the obstacle positions have to also be encoded in the state. The VIN was able to learn a policy for a general obstacle configuration by planning in a $m \times n$ state space by also taking into account the observation of the map.

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. The first convolution layer has 150 filters of size $(2m - 1) \times (2n - 1)$, which span the whole image and can convey information about the goal to every pixel. The second layer has 150 filters of size 1×1, and the third layer has 10 filters of size 1×1, to produce an output sized $10 \times m \times n$, similarly to the $\bar{Q}$ layer in our VIN. Similarly to the attention mechanism in the VIN, the values that correspond to the current state (pixel) are passed to a fully connected softmax output layer.

D.2 Mars Domain

We consider the problem of autonomously navigating the surface of Mars by a rover such as the Mars Science Laboratory (MSL) (Lockwood, 2006) over long-distance trajectories. The MSL has a limited ability for climbing high-degree slopes, and its path-planning algorithm should therefore avoid navigating into high-slope areas. In our experiment, we plan trajectories that avoid slopes of 10 degrees or more, using overhead terrain images from the High Resolution Imaging Science Experiment (HiRISE) (McEwen et al., 2007). The HiRISE data consists of grayscale images of the Mars terrain, and matching elevation data, accurate to tens of centimeters. We used an image of a 33.3km by 6.3km area at 49.96 degrees latitude and 219.2 degrees longitude, with a 10.5 sq. meters / pixel resolution. Each domain is a 128×128 image patch, on which we defined a 16×16 grid-world, where each state was considered an obstacle if its corresponding 8×8 image patch contained an angle of 10 degrees or more, evaluated using an additional elevation data. 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.

Our goal is to train a network that predicts the shortest-path trajectory *directly from the terrain image data*. We emphasize that the ground-truth elevation data *is not part of the input*, and the elevation therefore must be inferred (if needed) from the terrain image itself.

Our VIN design follows the model of Section 4.1. In this case, however, instead of feeding in the obstacle map, we feed in the raw terrain image, and accordingly modify the reward mapping f_R with 2 additional CNN layers for processing the image: the first with 6 kernels of size 5×5 and 4×4 max-pooling, and the second with a 12 kernels of size 3×3 and 2×2 max-pooling. The resulting $12 \times m \times n$ tensor is concatenated with the goal image, and passed to a third layer with 150 kernels of size 3×3 and a fourth layer with one 3×3 filter to output $\bar{R}$. The state inputs and output labels remain as in the grid-world experiments. We emphasize that the whole network is trained end-to-end, without pre-training the input filters.

In Table 4 we present our results for training a $m = n = 16$ map from a 10K image-patch dataset, with 7 random trajectories per patch, evaluated on a held-out test set of 1K patches. Figure 3 shows an instance of the input image, the obstacles, the shortest-path trajectory, and the trajectory predicted by our method. To put the 84.8% success rate in context, we compare with the best performance achievable without access to the elevation data. 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 network *did not* have access to these ground-truth obstacle classification labels during training or testing). Training this classifier is a standard binary classification problem, and its performance represents the best obstacle identification possible with our CNN in this domain. The best-achievable shortest-path prediction is then defined as the shortest path in an obstacle map generated by this classifier from the raw image. The results of this optimal predictor are reported in Table 1. The 90.3% success rate shows that obstacle identification from the raw image is indeed challenging. Thus, the success rate of the VIN network, which was trained without any obstacle labels, and had to 'figure out' the planning process is quite remarkable.

D.3 Continuous Control

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 [7], and Mujoco [30] for physical simulation. GPS works by learning time-varying iLQG controllers for each domain, and then fitting the controllers to a single NN policy using

	Pred. loss	Succ. rate	Traj. diff.
VIN	0.089	84.8%	0.016
Best achievable	-	90.3%	0.0089

Table 4: Performance of VINs on the Mars domain. For comparison, the performance of a planner that used obstacle predictions trained from labeled obstacle data is shown. This upper bound on performance demonstrates the difficulty in identifying obstacles from the raw image data. Remarkably, the VIN achieved close performance *without access* to any labeled data about the obstacles.

supervised learning. This process is repeated for several iterations, and a special cost function is used to enforce an agreement between the trajectory distribution of the iLQG and NN controllers. We refer to [17, 7] for the full algorithm details. For our task, we ran 10 iterations of iLQG, with the cost being a quadratic distance to the goal, followed by one iteration of NN policy fitting. This allows us to cleanly compare VINs to other policies without GPS-specific effects.

Our VIN design is similar to the grid-world cases: the state space $\bar{S}$ is a 16×16 grid-world, and the transitions $\bar{P}$ are 3×3 convolution kernels in the VI block, similar to the grid-world of Section 4.1. However, we made 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 (for example, the transitions for an action to go north-west would be γ in the top left corner and zeros otherwise), before training end-to-end. 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. The reward mapping f_R is a CNN with s_{image} as its input, one layer with 150 kernels of size 3×3, and a second layer with one 3×3 filter to output $\bar{R}$.

D.4 WebNav

"WebNav" [23] is a recently proposed goal-driven web navigation benchmark. In WebNav, web pages and links from some website form a directed graph $G(S, E)$. The agent is presented with a query text, which consists of N_q sentences from a target page at most N_h hops away from the starting page. The goal for the agent is to navigate to that target page from the starting page via clicking at most N_n links per page. Here, we choose $N_h = N_q = N_n = 4$. In [23], the agent receives a reward of 1 when reaching the target page via any path no longer than 10 hops. For evaluation convenience, in our experiment, the agent can receive a reward only if it reaches the destination via the *shortest path*, which makes the task much harder. We measure the top-1 and top-4 prediction accuracy as well as the average reward for the baseline [23] and our VIN model.

For every page s, the valid transitions are $A_s = \{s' : (s, s') \in E\}$.

For every web page s and every query text q, we utilize the bag-of-words model with pretrained word embedding provided by [23] to produce feature vectors $\phi(s)$ and $\phi(q)$. The agent should choose at most N_n valid actions from $A_s = \{s' : (s, s') \in E\}$ based on the current s and q.

The baseline method of [23] uses a single tanh-layer neural net parametrized by W to compute a hidden vector h: $h(s, q) = \tanh \left(W \begin{bmatrix} \phi(s) \\ \phi(q) \end{bmatrix} \right)$. The final baseline policy is computed via $\pi_{\text{bsl}}(s'|s, q) \propto \exp \left(h(s, q)^\top \phi(s') \right)$ for $s' \in A_s$.

We design a VIN for this task as follows. We firstly selected a smaller website as the approximate graph $\widetilde{G}(\bar{S}, \bar{E})$, and choose $\bar{S}$ as the states in VI. For query q and a page $\bar{s}$ in $\bar{S}$, we compute the reward $\bar{R}(\bar{s})$ by $f_R(\bar{s}|q) = \tanh \left((W_R \phi(q) + b_R)^\top \phi(\bar{s}) \right)$ with parameters W_R (diagonal matrix) and b_R (vector). For transition, since the graph remains unchanged, $\bar{P}$ is fixed. For the attention module $\Pi(\bar{V}^\star, s)$, we compute it by $\Pi(\bar{V}^\star, s) = \sum_{\bar{s} \in \bar{S}} \text{sigmoid} \left((W_\Pi \phi(s) + b_\Pi)^\top \phi(\bar{s}) \right) \bar{V}^\star(\bar{s})$, where W_Π and b_Π are parameters and W_Π is diagonal. Moreover, we compute the coefficient γ based on the query q and the state s using a tanh-layer neural net parametrized by W_γ: $\gamma(s, q) =$

Network	Top-1 Test Err.	Top-4 Test Err.	Avg. Reward
BSL	52.019%	24.424%	0.27779
VIN	50.562%	26.055%	0.30389

Table 5: Performance on the full wikipedia dataset.

$\tanh\left(W_\gamma \begin{bmatrix} \phi(s) \\ \phi(q) \end{bmatrix}\right)$. Finally, we combine the VI module and the baseline method as our VIN model by simply adding the outputs from these two networks together.

In addition to the experiments reported in the main text, we performed experiments on the full wikipedia, using 'wikipedia for schools' as the graph for VIN planning. We report our preliminary results here.

Full wikipedia website: The full wikipedia dataset consists 779169 training queries (3 million training samples) and 20004 testing queries (76664 testing samples) over 4.8 million pages with maximum 300 links per page.

We use the whole WikiSchool website as our approximate graph and set $K = 4$. In VIN, to accelerate training, we firstly only train the VI module with $K = 0$. Then, we fix $\bar{R}$ obtained in the $K = 0$ case and jointly train the whole model with $K = 4$. The results are shown in Tab. 5

VIN achieves 1.5% better prediction accuracy than the baseline. Interestingly, with only 1.5% prediction accuracy enhancement, VIN achieves 2.5% better success rate than the baseline: note that the agent can only success when making 4 consecutive correct predictions. This indicates the VI does provide useful high-level planning information.

D.5   Additional Technical Comments

Runtime: For the 2D domains, different samples from the same domain share the same VI computation, since they have the same observation. Therefore, a single VI computation is required for samples from the same domain. Using this, and GPU code (Theano), VINs are not much slower than the baselines. For the language task, however, since Theano doesn't support convolutions on graphs nor sparse operations on GPU, VINs were considerably slower in our implementation.

E   Hierarchical VI Modules

The number of VI iterations K required in the VIN depends on the problem size. Consider, for example, a grid-world in which the goal is located L steps away from some state s. Then, at least L iterations of VI are required to convey the reward information from the goal to state s, and clearly, any action prediction obtained with less than L VI iterations at state s is unaware of the goal location, and therefore unacceptable.

To convey reward information faster in VI, and reduce the effective K, we propose to perform VI at multiple levels of resolution. We term this model a hierarchical VI Network (HVIN), due to its similarity with hierarchical planning algorithms. In a HVIN, a copy of the input down-sampled by a factor of d is first fed into a VI module termed the high-level VI module. The down-sampling offers a $d\times$ speedup of information transmission in the map, at the price of reduced accuracy. The value layer of the high-level VI module is then up-sampled, and added as an additional input channel to the input of the standard VI module. Thus, the high-level VI module learns a mapping from down-sampled image features to a suitable reward-shaping for the nominal VI module. The full HVIN model is depicted in Figure 7. This model can easily be extended to include multiple levels of hierarchy.

Table 6 shows the performance of the HVIN module in the grid-world task, compared to the VIN results reported in the main text. We used a 2×2 down-sampling layer. Similarly to the standard VIN, 3×3 convolution kernels, 150 channels for each hidden layer $\bar{H}$ (for both the down-sampled image, and standard image), and 10 channels for the q layer in each VI block. Similarly to the VIN networks, the recurrence K was set relative to the problem size, taking into account the down-sampling factor: $K = 4$ for 8×8 domains, $K = 10$ for 16×16 domains, and $K = 16$ for 28×28 domains (in comparison, the respective K values for standard VINs were 10, 20, and 36). The HVINs demonstrated better performance for the larger 28×28 map, which we attribute to the improved information transmission in the hierarchical VI module.
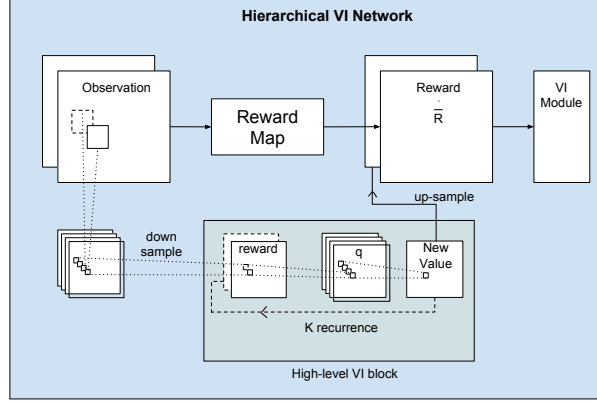
Figure 7: Hierarchical VI network. A copy of the input is first fed into a convolution layer and then downsampled. This signal is then fed into a VI module to produce a coarse value function, corresponding to the upper level in the hierarchy. This value function is then up-sampled, and added as an additional channel in the reward layer of a standard VI module (lower level of the hierarchy).

Domain	VIN			Hierarchical VIN		
	Prediction loss	Success rate	Trajectory diff.	Prediction loss	Success rate	Trajectory diff.
8×8	0.004	**99.6%**	0.001	0.005	99.3%	0.0
16×16	0.05	**99.3%**	0.089	0.03	99%	0.007
28×28	0.11	97%	0.086	0.05	**98.1%**	0.037

Table 6: HVIN performance on grid-world domain.