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.

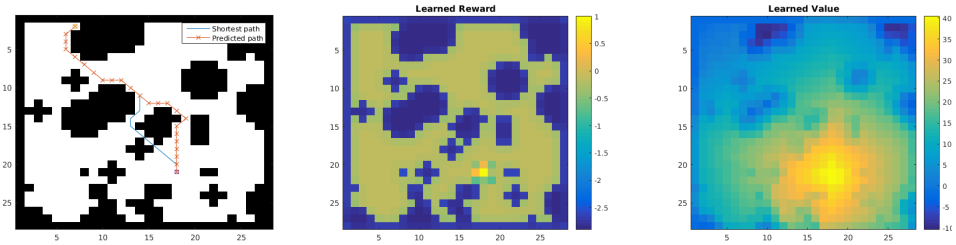

Figure 5: Visualization of learned reward and value function. Left: a sample domain. Center: learned reward f_R for this domain. Right: resulting value function (in VI block) for this domain.

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 [6].

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.

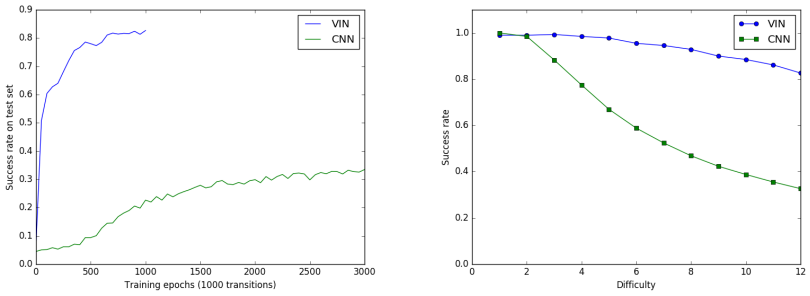

Figure 6: RL results – performance of VIN and CNN on 16×16 test maps. Left: Performance on all maps as a function of amount of training. Right: Success rate on test maps of increasing difficulty.

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 [28], implemented in the Theano [1] 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 [8], and Mujoco [29] for physical simulation. GPS works by learning time-

	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.

varying iLQG controllers for each domain, and then fitting the controllers to a single NN policy using 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, 8] 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 $\bar{G}(\bar{S}, \bar{E})$, 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 $f_R(\bar{s}|q) = \tanh\left(\left(W_R\phi(q) + b_R\right)^\top \phi(\bar{s})\right)$ 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 $\Pi(\bar{V}^\star, s) = \sum_{\bar{s} \in \bar{S}} \text{sigmoid}\left(\left(W_\Pi\phi(s) + b_\Pi\right)^\top \phi(\bar{s})\right) \bar{V}^\star(\bar{s})$, where W_Π and b_Π are parameters and W_Π is diagonal. Moreover, we compute the coefficient γ

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.

based on the query q and the state s using a tanh-layer neural net parametrized by W_γ: $\gamma(s,q) = \tanh\left(W_\gamma \begin{bmatrix} \phi(s) \\ \phi(q) \end{bmatrix}\right)$. 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 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

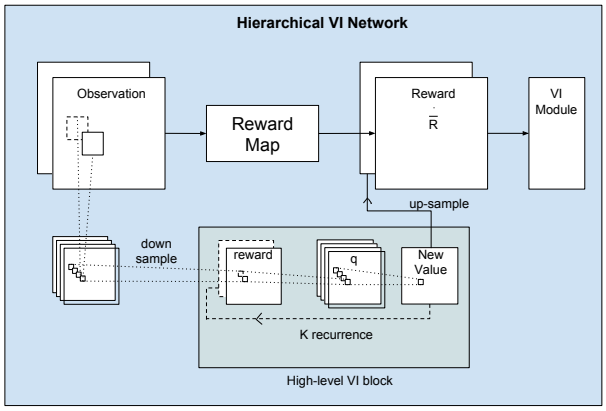

Figure 7: Hierarchical VI network. A copy of the input is first fed into a convolution layer and then downsampled. This signal is then fed into a VI module to produce a coarse value function, corresponding to the upper level in the hierarchy. This value function is then up-sampled, and added as an additional 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.

demonstrated better performance for the larger 28×28 map, which we attribute to the improved information transmission in the hierarchical VI module.