Value Iteration Networks

Aviv Tamar   AVIVT@BERKELEY.EDU
Sergey Levine   SLEVINE@EECS.BERKELEY.EDU
Pieter Abbeel   PABBEEL@CS.BERKELEY.EDU
University of California, Berkeley, Department of Electrical Engineering and Computer Sciences

Abstract

We introduce the *value iteration network*: a fully differentiable neural network with a 'planning module' embedded within. Value iteration networks are suitable for making predictions about outcomes that involve planning-based reasoning, such as predicting a desired trajectory from an observation of a map. 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 our value iteration networks on the task of predicting optimal obstacle-avoiding trajectories from an image of a landscape, both on synthetic data, and on challenging raw images of the Mars terrain.

Figure 1. Trajectory prediction problem on the Mars dataset. The input observation is a map of a terrain with obstacles, a starting position (blue marker), and a goal position (orange marker). The objective is to predict the shortest path to the goal between the obstacles.

1. Introduction

In recent years, deep convolutional neural networks (CNNs) have shown remarkable successes in computer vision tasks such as object recognition, action recognition, and scene labeling (Krizhevsky et al., 2012; Farabet et al., 2013; Ji et al., 2013). Broadly speaking, common to these tasks is the goal of answering the question: 'what is in the image ?' [1].

In this work, we consider prediction tasks with complex structured output that, broadly speaking, answer the question 'what to do given the image ?'. In particular, we are interested in predicting labels that are the result of some *planning* procedure. As an example, consider the path-planning problem depicted in Figure 1. Here, the observation is an image of a terrain and, given some start and goal positions, the label encodes the shortest-length trajectory between them. Such *structured prediction* tasks

---

[1] In general, the observation does not have to be an image, and CNNs have been applied to various inputs such as audio data and natural language text. In our presentation, we focus on image inputs, but our work can be extended to other types of observations.
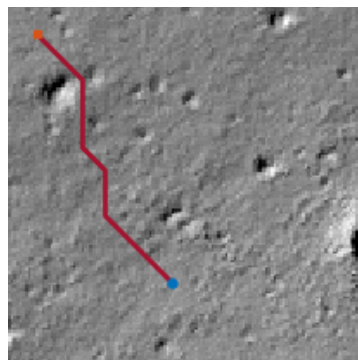
(Bakir et al., 2007; Nowozin & Lampert, 2011) are important for learning from demonstration in robotics (Hersch et al., 2008; Ratliff et al., 2009; Sung et al., 2015), activity forcasting (Kitani et al., 2012), and natural language processing (Chang et al., 2015).

A naïve approach for solving such problems is to ignore the underlying planning process altogether, and solve them using any standard multi-class classification model. For all but the smallest of problems, such an approach is doomed to fail, since the decisions made by the planning algorithm may have a very complex and non-smooth dependency on the observation, making generalization extremely difficult. For example, the trajectory in Figure 1 could change dramatically by adding just one more obstacle near its end.

In this work, we propose a neural network model that is suitable for learning such planning-based reasoning. Our model, termed a *value-iteration network*, has a 'planning module' embedded within the neural network structure. This embedding allows the network to implicitly encode an approximation of the planning process that generated the data, thereby enabling complex predictions that depend

on such planning.

The key to our approach is an observation that the classic value-iteration (VI) planning algorithm (Bellman, 1957; Bertsekas, 2012) may be represented by a specific type of convolutional neural network. By embedding such a VI network block inside a standard feed-forward classification network, we obtain a model that implicitly encodes a planning process. The VI block is differentiable, and the whole network can be trained *end-to-end* using standard backpropagation. After training, the network learns to map an observation to an underlying planning process, and generate predictions based on the resulting plan. In addition, we investigate a hierarchical extension of the VI block, which learns more efficiently on larger domains.

We show that value-iteration networks can learn to predict non-trivial trajectories from images of synthetic grid-world domains, and directly from raw overhead images of Mars terrain, as in Figure 1. Our approach performs significantly better than feed-forward classification networks, and does not require knowing the planning model in advance.

We emphasize that our contribution is a new *representation* for planning-based predictions, and not a new learning algorithm. Indeed, in this work we used standard supervised learning methods for training our models. Previous approaches to the structured-prediction problem, such as inverse reinforcement learning (IRL) address the underlying planning process in the learning algorithm (Abbeel & Ng, 2004; Ratliff et al., 2006; Ziebart et al., 2008; Ross et al., 2011). Our contribution is therefore complimentary to these methods, and can be easily incorporated within more advanced learning algorithms, such as those proposed by Ross et al. (2011). We further discuss related work in Section 5.

2. Background

In this section we provide background on planning and convolutional neural networks, and present our problem formulation.

2.1. Value Iteration

A standard model for sequential decision making and planning is the Markov decision process (MDP) (Bellman, 1957; Bertsekas, 2012). An MDP 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,

$$V^\pi(s) \doteq \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s \right],$$

where $\gamma \in (0,1)$ is a discount factor, and $\mathbb{E}^\pi$ denotes an expectation over trajectories of states and actions $s_0, a_0, s_1, a_1 \ldots$, in which actions are selected according to 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):

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

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 $\pi^*(s) = \arg\max_a Q_\infty(s,a)$. In practice, VI can only be applied for a finite number of iterations, yielding an approximate solution, for which error bounds can be derived (Bertsekas, 2012).

2.2. Convolutional Neural Networks

Convolutional neural networks (CNNs; LeCun et al. 1998; Bengio 2009) are neural networks with a particular architecture that has proved useful for computer vision, among other domains. A CNN is comprised of stacked convolution and max-pooling layers. The input to each convolution layer is a 3-dimensional signal X, typically, an image with l channels, m horizontal pixels, and n vertical pixels, and its output h is a l'-channel convolution of the image with kernels $W^1, \ldots, W^{l'}$,

$$h_{l',i',j'} = \sigma \left(\sum_{l,i,j} W_{l,i,j}^{l'} X_{l,i'-i,j'-j} \right), \quad (2)$$

where σ is a non-linear activation function, typically a rectified linear unit (ReLU): $\sigma(x) = \max(x,0)$. A max-pooling layer selects, for each channel l and pixel i,j in h, the maximum value among its neighbors $N(i,j)$

$$h_{l,i,j}^{maxpool} = \max_{i',j' \in N(i,j)} h_{l,i',j'}. \quad (3)$$

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), and the gradients of a CNN can be calculated efficiently using backpropagation.

2.3. Problem Formulation

We consider a structured prediction (Bakir et al., 2007; Nowozin & Lampert, 2011) problem in which the input consists of a dataset of images $\left\{x^i\right\}_{i=1,\ldots,N}$, states $\left\{s^i\right\}_{i=1,\ldots,N}$, and action labels $\left\{a^i\right\}_{i=1,\ldots,N}$. We assume that the actions are generated from an optimal policy $a^i = \pi^*(s^i)$ with respect to some unknown MDP, encoded in the image x^i. For example, in the path planning domains we consider, the state s encodes the position, and the image shows the obstacles and a goal location. The action then encodes the direction of the shortest path to the goal between the obstacles. We emphasize that the reward and transitions in the MDP are unknown, and are not an explicit part of the input in any way.

Our goal is to learn a mapping $y(x, s)$ that best predicts the correct action label for an image and state. Naïvely, this problem can be cast as a standard supervised learning problem (Bishop, 2006), by concatenating x and s into a single observation. With enough training data, it is conceivable that a standard supervised learning algorithm would learn a suitable solution. However, such a naïve approach ignores the knowledge about the process that generates the data. By exploiting this knowledge, a solution may be found more efficiently. In the following, we propose a class of neural network representations that incorporate a 'planning module' in their structure. We shall show that this structure enables much more efficient learning than standard feedforward networks in this problem domain.

3. The Value Iteration Network Model

In this section we introduce the value iteration network – a model that encodes a differentiable planning procedure.

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 can be performed.

Following this idea, we propose the VI network block model, as depicted in Figure 2. The inputs to the VIN are a 'reward image' r of dimensions l, m, n, and a state $(i_s, j_s) \in (1,\ldots,m) \times (1,\ldots,n)$. Thus, we explicitly assume that the state space maps to a 2-dimensional map[2].

---

[2]Our approach can be extended to general state spaces by adding a transformation layer between the state and the value function (Jaderberg et al., 2015). This extension will be explored

The reward is fed into to a convolutional layer q with A channels and a linear activation function,

$$q_{a,i',j'} = \sum_{l,i,j} W^a_{l,i,j} r_{l,i'-i,j'-j}.$$

Each channel in this layer corresponds to $Q(s, a)$ for a particular action a. This layer is then max-pooled along the actions channel to produce the next-iteration value function layer v,

$$v_{i,j} = \max_a q(a, i, j).$$

The next-iteration value function layer v is then stacked with the reward, and fed back into the convolutional layer and max-pooling layer K times, to perform K iterations of value iteration. After K such recurrences, the A channels of the q layer in the i_s, j_s position are fed into a fully connected softmax output layer y,

$$y(a) = \frac{\exp\left(\sum_{a'} W^{out}_{a',a} q(a', i_s, j_s)\right)}{\sum_a \exp\left(\sum_{a'} W^{out}_{a',a} q(a', i_s, j_s)\right)}.$$
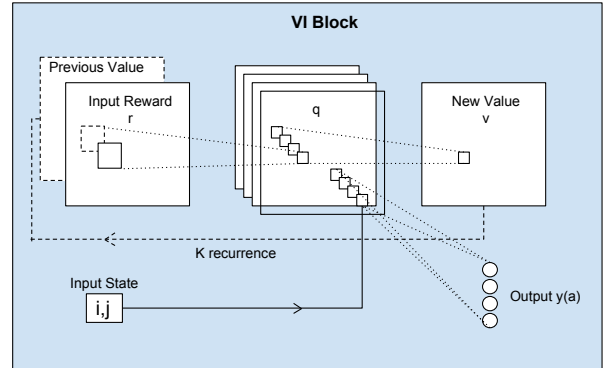

Figure 2. The VI block. A neural-network that approximates the VI algorithm.

The VI block is simply a neural-network based module for approximate-VI based planning. Nevertheless, we emphasize that this representation makes *learning* the MDP parameters and reward function natural – by backpropagating through the network, similarly to a standard CNN. Thus, the weights in a VI block, which implicitly encode the MDP transitions and reward, need not be known in advance, and are trained *directly from data*. This makes the VI block a suitable model for learning action predictions that are the output of some planning process, as in the trajectory prediction problems described above. We additionally emphasize that the weights learned by the VI block can be very different from the true MDP transitions that generated the data, so long as they lead to similar action

---

in future work.

predictions. In particular, the number of channels A in the q layer can be different than the true number of actions in the true MDP. Indeed, in our experiments we found out that selecting a larger A than the number of actions in the true MDP actually improves learning.

The VI block can be connected to an image-processing network that takes as input a general image, and outputs a reward map. The key here is that *the whole network can be trained end-to-end*, by back-propagating gradient information through the network components. Thus, the image-processing network can be trained to learn a *mapping from the image features to a reward map* representing the planning process underlying the data. We emphasize that this learned reward mapping need not be the true MDP reward, so long as it leads to accurate action predictions. In this sense, the the network is learning a *reward shaping* (Ng et al., 1999) from data. In Figure 3 we depict a VI network with two additional convolution layers between the image input and the VI block. This specific network was used in our experiments, but other image-processing models can be easily incorporated instead.
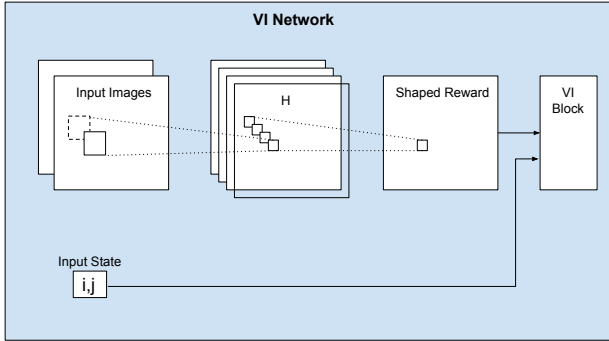

Figure 3. VI network. The input observation is processed by two standard convolution layers, and fed into a VI block as a reward input.

3.1. Hierarchical VI Networks

The difficulty of training deep recurrent models such as the VI block described above increases with the number of VI iterations K (Pascanu et al., 2012). Unfortunately, for some planning domains, the number of VI iteration required to obtain a reasonable policy may be quite large. As an example, consider 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.

At least potentially, a well-chosen reward shaping can help mitigate this issue. However, for a localized mapping from

the input image to the shaped reward, such as the CNN layers in the VI network suggested above, the reward-information problem may pertain.

Our solution for conveying reward information faster in VI is 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 block termed the high-level VI block. 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 block is then up-sampled, and added as an additional input channel to the shaped reward of the standard VI network. Thus, the high-level VI block learns a mapping from down-sampled image features to a suitable reward-shaping for the nominal image. The full HVIN model is depicted in Figure 4. This model can easily be extended to include multiple levels of hierarchy.
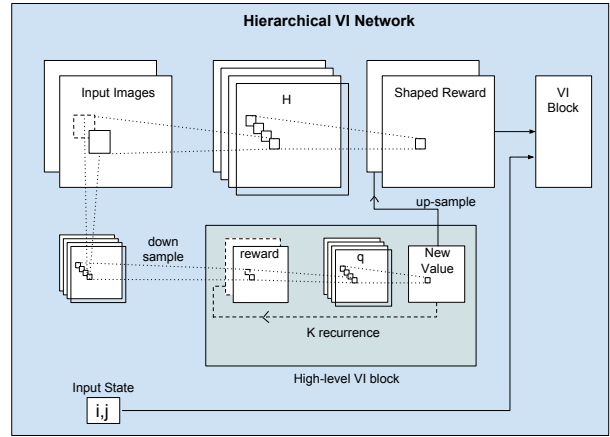

Figure 4. Hierarchical VI network. A copy of the input is first fed into a convolution layer and then downsampled. This signal is then fed into a VI block to produce a coarse value function, corresponding to the upper level in the hierarchy. This value function is then up-sampled, and added as an additional channel in the reward layer of a standard VI network (lower level of the hierarchy).

3.2. Approximate Planning Interpretation

Our presentation so far has been deep-learning oriented, viewing approximate VI as a neural-network architecture. In this section we provide a different interpretation, and discuss VI networks as a planning algorithm.

In this context, the VI network can be seen as an *approximate* value iteration algorithm, with the network weights approximating the MDP transitions, and the shaped reward function replacing the true reward. Most importantly, in this approximation VI is not iterated until convergence,

but only for K iterations. In the planning literature, several studies investigated the use of reward-shaping (Ng et al., 1999), hierarchical macro-actions (Mann & Mannor, 2014), and state-aggregation (Taylor et al., 2009; Bertsekas, 2012) for reducing the number of iterations required for VI to converge, with the goal of reducing computation time. However, while it is known that a well-chosen reward-shaping, state-aggregation, or macro-action set can indeed make VI converge faster, the problem of choosing such is challenging. While several useful heuristic rules and intractable optimality-driven approaches exist, to our knowledge, a principled yet tractable method is not yet known.

One interpretation of our VI networks is that, by training from optimal actions, we are in fact learning an optimal reward-shaping for the iteration-constrained VI. More interestingly, in the HVIN models, we are also learning to perform a hierarchical VI algorithm. In this algorithm, the network performs state-aggregation (by down-sampling), with the actions in the high-level VI block representing macro-actions in the original domain. Interestingly, the macro-actions here are learned *from data*. We are not aware of an analogous hierarchical VI algorithm in the dynamic programming literature that does not rely on hand-crafted macro-actions.

At least conceptually, our method may be used in the future to devise reward-shaping and macro-action discovery techniques for planning algorithms.

4. Results

We evaluated the VIN model in two path-planning domains. The first domain is a synthetic grid-world with obstacles, in which the observation is a map of the obstacles and the goal position. The second domain models a navigation task for a Mars rover. In this domain, the observation is an overhead image of the terrain, along with a map of the goal position. Our goal in these experiments is two-fold. First, we wish to show that VIN models are vastly superior to standard feed-forward networks in prediction tasks that involve planning. Second, we shall demonstrate that by embedding a VIN within an image-processing network, challenging structured-prediction tasks from real images can be tackled successfully.

4.1. Synthetic Grid-World Domain

We consider a $m \times n$ grid-world with randomly placed obstacles, and a border of size 1. Figure 5 shows an instance of the grid world with $m = n = 16$. The possible actions are moving in one of the 8 directions. Attempting to move into an obstacle results in no movement. Finding a shortest path between two locations in the grid-world may be
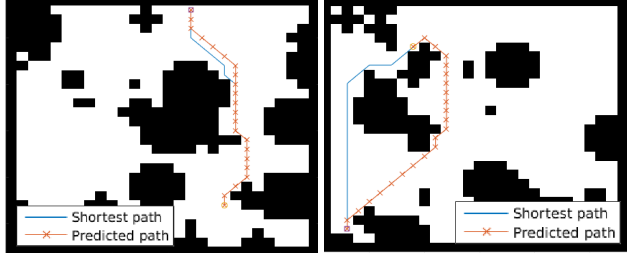

Figure 5. Grid-world domain. Two random instances of the 28 × 28 map are shown, with the path predicted by the HVIN network, and the ground-truth shortest path trajectory. In the left domain, the trajectories mostly overlap. Note that the shortest-path trajectory is not unique, and in the left domain the trajectory predicted by the HVIN is different than the ground-truth, but of the same length.

done using value iteration (among other standard planning algorithms), by using a constant negative reward for every non-goal state, and some positive reward for the goal.

Recall that our goal in the experiment is to solve a classification problem – to learn a mapping from an image of the grid-world (as in Figure 5), the current position in the map, and some goal position, to an action that leads to the shortest path to the goal. Our training set consists of $N_i = 5000$ random grid-world instances, with $N_t = 7$ shortest-path trajectories 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 X. The first channel of X encodes the obstacle presence (1 for obstacle, 0 otherwise), while the second channel encodes the goal position (1 at the goal, 0 otherwise). 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 trained several neural-network based multi-class logistic regression classifiers using stochastic gradient descent, with an RMSProp step size (Tieleman & Hinton, 2012), implemented in the Theano (Bastien et al., 2012) library.

We compare between the following neural network models:

VIN network We used the VIN model of Section 3, with 3×3 convolution kernels, 150 channels for the hidden layer H, and 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, $K = 36$ for 28×28 domains, and $K = 44$ for 36×36 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.

Hierarchical VIN network We used the hierarchical VIN model of Section 3.1, with a 2×2 down-sampling layer. Similarly to the VI network, we used 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. Similary 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, $K = 16$ for 28×28 domains, and $K = 20$ for 36×36 domains.

Fully-connected feed-forward network In order to train a standard multi-layer fully-connected (FC) feed-forward network (also known as a multi-layer perceptron model), we flattened out the obstacle and goal images, and represented the state (i, j) as two 1-hot vectors (a vector for each coordinate), for a total of $2 \times m \times n + m + n$ inputs. We constructed both a single hidden-layer FC network, with 256 hidden units, and a two-layer FC network, with 256 hidden units in each hidden layer, and ReLU non-linearity. The number of hidden units was chosen by trial and error.

Feed-forward CNN This model is an extension of the single layer FC network, in which the input images are first fed into 2 convolution layers, with 3×3 convolution kernels, 8 and 16 channels, respectively, and 2×2 max-pooling. The output of these layers, along with the 1-hot state vectors, are fed into a single hidden-layer FC network, with 256 hidden units.

In Table 1 we present the average $0 - 1$ prediction loss of each model, evaluated on a held-out test-set. In addition, we report the actual performance in predicting a trajectory by measuring the average success rate in obstacle avoidance, and the length of the trajectory compared to the optimal shortest-path. We evaluated these measures on a held-out test-set of 1000 maps with random obstacles, goal positions, and initial states. For each map, a full trajectory from the initial state was predicted, by iteratively rolling-out the next-states predicted by the network. A trajectory was said to succeed if it did not hit any 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, the VIN and hierarchical VIN significantly outperform the feed-forward networks. Note, however, that the performance gap increases dramatically with the problem size. The 8×8 domain is small enough for a feed-forward network to 'remember' all possible obstacle configurations, and perform reasonably well. The number of possible configurations, however, increases exponentially with the problem size, leading to a complete failure on the larger maps. The VIN based models, on the other hand, are able to overcome this issue by exploiting the *structure* of the problem.

Note that the hierarchical VIN has a better performance on the larger maps, even though it uses a smaller number of VI iterations K. Moreover, the values of K for hierarchical VIN were chosen to be smaller than the image size, making it impossible to predict correct labels far away from the goal without exploiting hierarchical information. Thus, the near-perfect performance of hierarchical VIN shows that the network indeed learned a useful reward-shaping from the down-sampled VI block.

We emphasize that the model (i.e., transition dynamics) used for planning the shortest-path trajectories was not given to the learning algorithm at any point. In such a setting, exact IRL is impossible.

4.2. Mars Rover Navigation

The grid-world experiments in the previous section establish the capability of VIN networks to efficiently represent shortest-path predictions from image input. One very simplifying factor in those experiments, however, was that the image input was a prefect map of the planning domain. In this section we show that by piping VIN networks to a CNN feature extraction block, and training the whole network *end-to-end*, we can learn relevant features and make shortest-path predictions *from raw image input*. We demonstrate this on predicting obstacle-avoidance trajectories directly from overhead terrain images of a Mars landscape.

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 digital terrain models 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. From this image, we randomly sampled $8m$ pixel by $8n$ pixel patches, and planned shortest-path trajectories on a $m \times n$ 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 the ground-truth elevation data. Thus, the MDP underlying shortest-path planning in this case is similar to the grid-world domain of Section 4.1. An example of the domain and terrain image is depicted in Figure 6.

Domain	VIN			Hierarchical VIN			1-layer FC		
	Prediction loss	Success rate	Trajectory diff.	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**	0.08	80.6%	0.001
16×16	0.05	**99.3%**	0.089	**0.03**	99%	**0.007**	0.25	54.4%	0.14
28×28	0.11	97%	0.086	**0.05**	**98.1%**	**0.037**	0.40	7.1%	0.80
36×36	0.14	92.3%	0.21	**0.09**	**93.8%**	**0.14**			

Domain	2-layer FC			CNN + 1-layer FC		
	Prediction loss	Success rate	Trajectory diff.	Prediction loss	Success rate	Trajectory diff.
8×8	0.08	82.8%	0.004	0.08	78.5%	0.02
16×16	0.24	55.8%	0.07	0.22	19.3%	0.76
28×28	0.35	22.1%	0.62			

Table 1. Performance on grid-world domain. For all domain sizes, VIN networks significantly outperform standard feed-forward networks. Note that the performance gap increases dramatically with problem size. In addition, hierarchical VIN outperforms standard VIN on the larger domains.

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 network model is the hierarchical VIN model of Section 3.1, with a 2×2 down-sampling layer. In this case, however, instead of feeding in the obstacle map, we feed in the raw terrain image. This image is processed by two convolution layers: the first with a 5×5 kernel, 6 channels, and 4×4 max-pooling, and the second with a 3×3 kernel, 12 channels, and 2×2 max-pooling. The resulting 12-channel $m \times n$ image is fed into the VI network instead of the obstacle map. The goal map, 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 2 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. 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

	Prediction loss	Success rate	Trajectory diff.
Hierarchical VIN	0.089	84.8%	0.016
Best achievable	-	90.3%	0.0089

Table 2. Performance of the HVIN on the Mars domain. For comaprison, the performance of a trajectory planner that used obstacle predictions trained from labeled obstacle data is shown. This upper bound on performance demonstrates the challenge in accurately identifying obstacles from the raw image data. Remarkably, the HVIN achieved close performance *without access* to any labeled data about the obstacles.

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.

In Figure 6 we show an instance of the input image, the obstacles, the shortest-path trajectory, and the trajectory predicted by our method.

5. Related Work

Learning to predict trajectories is an instance of structured prediction problems (Bakir et al., 2007; Nowozin & Lampert, 2011). In particular, structured prediction strategies such as maximum-margin prediction (Ratliff et al., 2006) have been applied to imitation learning of trajectories from demonstrations. In this setting, the transitions in the MDP underlying the planning process are assumed to be known,
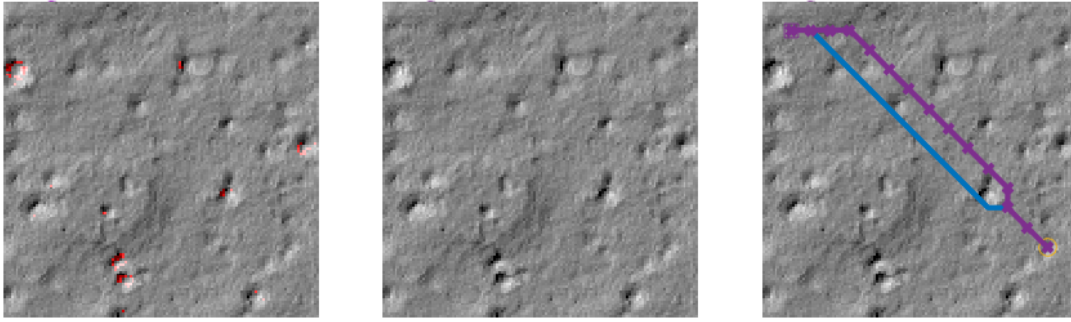
Figure 6. Mars domain (best viewed in color). Left: a sample 128 by 128 image patch from the HiRISE photography of the Mars terrain data, with points of elevation sharper than $10°$ colored in red. These points were calculated from a matching image of elevation data (not shown), and were not available to the learning algorithm. Middle: the actual image that was used for training the network. Note the difficulty of distinguishing between obstacles (red dots in the left figure) and non-obstacles. Right: the trajectory predicted by the trained HVIN, between a random starting and goal positions (purple line, with cross markers), and the shortest-path ground truth trajectory between these points (blue line).

but the cost has to be learned from data. A similar approach is also used in inverse reinforcement learning (IRL) (Abbeel & Ng, 2004; Ziebart et al., 2008; Levine et al., 2011), which has been used for similar domains of predicting trajectories from overhead images as in this work (Kitani et al., 2012; Ratliff et al., 2009). Our approach is somewhat orthogonal to these methods, as we encode the planning process directly in our policy, relieving the need to use specialized learning algorithms that exploit the underlying MDP. To emphasize this point, in this work we trained our models using a standard classification-learning algorithm. However, our approach can be easily incorporated within a no-regret structured prediction method such DAgger (Ross et al., 2011) to potentially yield improved results. We remark that although we do not explicitly use a model, the local convolutions applied in our networks do exploit some structure of the underlying MDP. However, this prior knowledge is still considerably less than knowing the exact transitions.

Similar to our approach, Zucker & Bagnell (2012) suggested gradient-based approaches for learning better planners. In that work, a linearly parameterized cost function was learned using reinforcement-learning techniques. In a different study, Singh et al. (2010) used evolutionary algorithms to learn a cost function that accelerates learning. Our work is substantially different, in learning both costs and transitions of an approximate planning algorithm, and using highly expressive neural networks instead of linear function approximators.

Another relevant line of work is the recent trend of approximating algorithms using neural networks, such as the neural turing machines (Graves et al., 2014), pointer net-

works (Vinyals et al., 2015), and the recurrent networks of Zaremba et al. (2015). Our approach can be seen as a neural-network approximation of dynamic programming (VI), learned from examples.

6. Conclusion and Outlook

We introduced the value-iteration network: a fully differentiable neural network with a 'planning module' embedded within. VI networks are particularly well-suited for making predictions about the outputs of a planning process, for example, trajectory planning, as explored in this work. By training a VI end-to-end, we were able to learn to predict shortest-path trajectories directly from raw images of a domain, both on synthetic data, and real overhead images of a Mars terrain. In addition, we proposed a hierarchical variant of the VI network, which is better suited for larger domains.

In this work we focused on planning problems that have a two-dimensional structure, in which the state can be easily mapped onto images, and the transitions are local. While such problems have important applications in activity forecasting and path-planning (Kitani et al., 2012; Ratliff et al., 2009), generalizing our approach to different problem structures is important. We now discuss several directions for achieving this. First, the mapping from the state input to the shaped-reward map can be generalized by adding a spatial transformation layer (Jaderberg et al., 2015) between them. Such a transformation can be learned simultaneously with the rest of the network by training end-to-end. Second, the localized convolution kernels, which imply local state transitions in the MDP, can easily be replaced with different, arbitrary, kernel shapes. In addition,

conditioning the kernels on the input (Oh et al., 2015) may provide an additional flexibility.

We conclude with an alternative view of our approach in a reinforcement learning (RL) context. The VI network can be seen as a policy representation, i.e., a mapping from states to actions, that explicitly performs some look-ahead computation by the VI block. Recent RL advances (Mnih et al., 2015; Schulman et al., 2015) showed that CNN-based policies can be trained successfully in a RL setting. Thus, with minimal modification, our VI networks may be incorporated into a RL algorithm, with the potential of providing stronger policy representations.

References

Abbeel, P. and Ng, A. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

Bakir, G. H, Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., and Vishwanathan, S. *Predicting Structured Data (Neural Information Processing)*. The MIT Press, 2007.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., and Bengio, Y. Theano: new features and speed improvements. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2012.

Bellman, R. *Dynamic Programming*. Princeton University Press, 1957.

Bengio, Y. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

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

Bishop, C. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN 978-0387-31073-2.

Chang, K., Krishnamurthy, A., Agarwal, A., Daumé III, H., and Langford, J. Learning to search better than your teacher. In *ICML*, 2015.

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

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Hersch, M., Guenter, F., Calinon, S., and Billard, A. Dynamical system modulation for robot learning via kinesthetic demonstrations. *Robotics, IEEE Transactions on*, 24(6):1463–1467, 2008.

Jaderberg, M., Simonyan, K., and Zisserman, A. Spatial transformer networks. In *NIPS*, 2015.

Ji, S., Xu, W., Yang, M., and Yu, K. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

Kitani, K., Ziebart, B., Bagnell, A., and Hebert, M. Activity forecasting. *European Conference on Computer Vision*, 2012.

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

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

Levine, S., Popovic, Z., and Koltun, V. Nonlinear inverse reinforcement learning with gaussian processes. In *NIPS*, 2011.

Mann, T. and Mannor, S. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *ICML*, 2014.

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

Ng, A., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

Nowozin, S. and Lampert, C. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

Oh, J., Guo, X., Lee, H., Lewis, R., and Singh, S. Action-conditional video prediction using deep networks in atari games. In *NIPS*, 2015.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

Ratliff, N., Bagnell, A., and Zinkevich, M. Maximum margin planning. In *ICML*, 2006.

Ratliff, N., Silver, D., and Bagnell, A. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

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

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

Singh, S., Lewis, R., Barto, A., and Sorg, J. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

Sung, J., Jin, S., and Saxena, A. Robobarista: Object part-based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *International Symposium on Robotics Research (ISRR)*, 2015.

Taylor, J., Precup, D., and Panagaden, P. Bounding performance loss in approximate MDP homomorphisms. In *NIPS*, 2009.

Tieleman, T. and Hinton, G. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NIPS*, 2015.

Zaremba, W., Mikolov, T., Joulin, A., and Fergus, R. Learning simple algorithms from examples. *arXiv preprint arXiv:1511.07275*, 2015.

Ziebart, B., Maas, A., Bagnell, A., and Dey, A. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

Zucker, M. and Bagnell, A. Reinforcement planning: RL for optimal planners. In *International Conference on Robotics and Automation (ICRA)*, 2012.