Value Iteration Networks

Aviv Tamar[1], **Yi Wu**[1], **Garrett Thomas**[1], **Sergey Levine**[1], **Pieter Abbeel**[1,2]
[1]UC Berkeley, [2]OpenAI
{avivt,jxwuyi,gwthomas}@berkeley.edu, svlevine@eecs.berkeley.edu, pabbeel@cs.berkeley.edu

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.

This paper is a significantly abridged and IJCAI audience targeted version of the original NIPS 2016 paper with the same title, available here: `https://arxiv.org/abs/1602.02867`

1 Introduction

Over the last decade, deep convolutional neural networks (CNNs) have revolutionized supervised learning for object recognition [Krizhevsky *et al.*, 2012], among other computer vision tasks. Recently, CNNs have been applied to reinforcement learning (RL) tasks with visual observations such as Atari games [Mnih *et al.*, 2015], robotic manipulation [Levine *et al.*, 2016], and imitation learning (IL) [Giusti and others, 2016]. 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* [Bertsekas, 2012]. However, most recent deep RL works [Mnih *et al.*, 2015; Levine *et al.*, 2016; Giusti and others, 2016] 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.

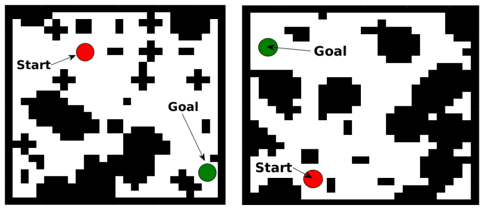

Figure 1: Two instances of a grid-world domain. Task is to move to the goal between the obstacles.

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.

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 [Bellman, 1957] 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.

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. 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

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) [Bertsekas, 2012]. 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, $V^\pi(s) \doteq \mathbb{E}^\pi\left[\sum_{t=0}^\infty \gamma^t r(s_t,a_t)\middle| s_0=s\right]$, where $\gamma \in (0,1)$ is a discount factor, and $\mathbb{E}^\pi$ denotes an expectation over trajectories of states and actions $(s_0, a_0, s_1, a_1 \dots)$, 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):

$$V_{n+1}(s) = \max_a Q_n(s,a) \quad \forall s, \tag{1}$$

where $Q_n(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_n(s')$. 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 [Fukushima, 1979; Krizhevsky *et al.*, 2012]. 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 [Ross *et al.*, 2011; Mnih *et al.*, 2015; Schulman *et al.*, 2015; Levine *et al.*, 2016] 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 $\left\{\phi(s^i), a^i \sim \pi^*(\phi(s^i))\right\}_{i=1,\dots,N}$ is generated by an expert. Learning a policy then becomes an instance of supervised learning [Ross *et al.*, 2011]. 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 [Sutton and Barto, 1998; Mnih *et al.*, 2015; Schulman *et al.*, 2015; Levine *et al.*, 2016] use these observations to improve the value of the policy.

3 The Value Iteration Network Model

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 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* [Xu *et al.*, 2015], 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.

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 main observation is that each iteration of VI (1) 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 used in our natural-language experiments. 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.

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 . Thus, a rough design can be specified, and then tuned by end-to-end training.

4 Experiments

We evaluated VINs as policy representations on various domains. The full results, implementation details, and source code are presented in [Tamar *et al.*, 2016]; here we provide a brief summary.

Grid-World Domain This domain is a synthetic grid-world with randomly placed obstacles, where the observation includes the position of the agent, and also an image of the map of obstacles and goal position. 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 [Tamar *et al.*, 2016], we report additional RL experiments that show similar findings.

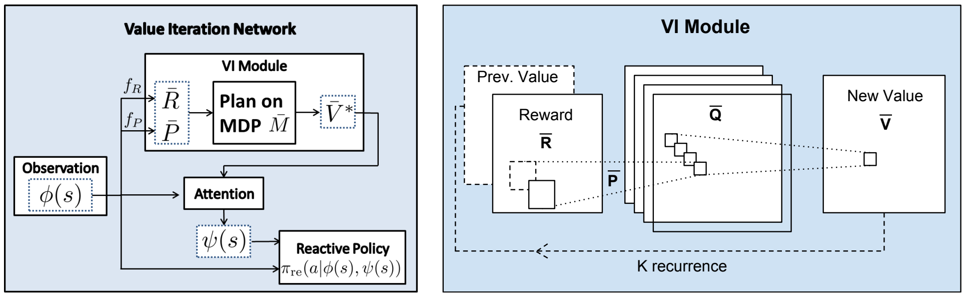

Figure 2: Planning-based NN models. Left: a general policy representation that adds value function features from a planner to a reactive policy. Right: VI module – a CNN representation of VI algorithm.

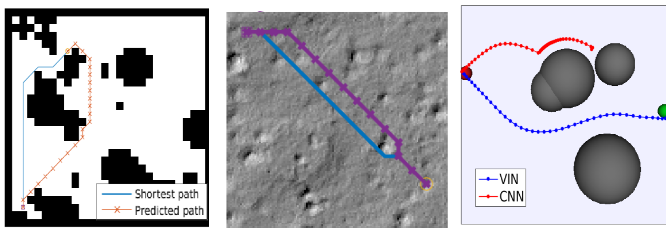

Figure 3: Experiments (best viewed in color). Left: a random instance of the 28 × 28 gridworld, with VIN-predicted trajectory and ground-truth shortest path. Middle: Navigation with natural image input of a Mars terrain – VIN-predicted (purple, cross markers) and ground truth (blue) trajectories. Right: Continuous control domain – comparing a VIN-predicted trajectory with a CNN policy baseline.

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 . 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: (a) a **CNN**-based reactive policy inspired by the recent impressive results of DQN [Mnih *et al.*, 2015], and (b) a **Fully Convolutional Network (FCN)** [Long *et al.*, 2015], in which each pixel in the image is assigned a semantic label - the action. In Table 1 we present the average prediction error of the correct action (evaluated on a held-out test-set of randomly-generated maps) for different problem sizes. In addition, we evaluate the success rate – the probability of successfully reaching the goal without hitting obstacles. VINs 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,

Domain	VIN		CNN		FCN	
	Pred. loss	Succ. rate	Pred. loss	Succ. rate	Pred. loss	Succ. rate
8×8	0.004	**99.6%**	0.02	97.9%	0.01	97.3%
16×16	0.05	**99.3%**	0.10	87.6%	0.07	88.3%
28×28	0.11	**97%**	0.13	74.2%	0.09	76.6%

Table 1: Performance on grid-world domain. For all domain sizes, VIN networks significantly outperform standard reactive networks. Note that the performance gap increases dramatically with problem size.

and learn the easily predictable parts of the trajectory yet fail on the complete task.

Additional Domains VIN architectures can be easily composed with other NN architectures. We demonstrated this idea by composing VINs with NNs for perception and control, and applied the resulting architectures to a navigation task using overhead images of a Mars terrain, and also to a continuous control of navigating between obstacles. In addition, we applied VINs to WebNav [Nogueira and Cho, 2016] – a language based search task on a graph. We refer to [Tamar *et al.*, 2016] for the full details. The main observation from our experiements is that VINs generalize better than reactive policies to tasks that were not seen during training.

5 Conclusion and Outlook

While deep RL has gained much interest recently, few 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 [Sutton and Barto, 1998; Mnih *et al.*, 2015; Duan *et al.*, 2016]. 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. In future work we intend to learn different planning computations, based on simulation [Guo *et al.*, 2014], or optimal linear control [Watter *et al.*, 2015], and combine them with reactive policies, to potentially develop RL solutions for task and motion planning [Kaelbling and Lozano-Pérez, 2011].

Acknowledgments

Research partially funded by Siemens, ONR PECASE award, Army Research Office through the MAST program, and an NSF CAREER award (#1351028). A. T. partially funded by the Viterbi Scholarship, Technion. Y. W. partially funded by a DARPA PPAML program, contract FA8750-14-C-0011.

References

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

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

[Ciresan *et al.*, 2012] 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.

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

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

[Farabet *et al.*, 2013] 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.

[Finn *et al.*, 2016] 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.

[Fukushima, 1979] K. Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position- neocognitron. *Transactions of the IECE*, J62-A(10):658665, 1979.

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

[Guo *et al.*, 2014] 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.

[Guo *et al.*, 2016] 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.

[Ilin *et al.*, 2007] 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.

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

[Kaelbling and Lozano-Pérez, 2011] 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.

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

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

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

[Long *et al.*, 2015] 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.

[Mnih *et al.*, 2015] 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.

[Mnih *et al.*, 2016] 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.

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

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

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

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

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

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

[Tamar *et al.*, 2016] A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel. Value iteration networks. In *NIPS*, 2016.

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

[Todorov *et al.*, 2012] 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.

[Watter *et al.*, 2015] 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.

[Xu *et al.*, 2015] 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.