# Integrating Algorithmic Planning and Deep Learning for Partially Observable Navigation

Peter Karkus[1,2], David Hsu[1,2] and Wee Sun Lee[2]

***Abstract*— We propose to take a novel approach to robot system design where each building block of a larger system is represented as a differentiable program, i.e. a deep neural network. This representation allows for integrating algorithmic planning and deep learning in a principled manner, and thus combine the benefits of model-free and model-based methods. We apply the proposed approach to a challenging partially observable robot navigation task. The robot must navigate to a goal in a previously unseen 3-D environment without knowing its initial location, and instead relying on a 2-D floor map and visual observations from an onboard camera. We introduce the Navigation Networks (NavNets) that encode state estimation, planning and acting in a single, end-to-end trainable recurrent neural network. In preliminary experiments we successfully trained navigation networks to solve the challenging partially observable navigation task in simulation.**

## I. Introduction

Humans employ two primal approaches to decision-making: reasoning with models and learning from experiences. The former is called planning, and the latter, learning. By integrating planning and learning methods, AlphaGo has recently achieved super-human performance in the challenging game of Go [1]. Robots must also integrate planning and learning to address truly difficult decision-making tasks, and ultimately to achieve human-level robotic intelligence.

Robots act in the real world that is inherently uncertain and only partially observable. Under partial observability the robot cannot determine its state exactly. Instead, it must integrate information over the past history of its actions and observations. Unfortunately, this drastically increases the complexity of decision making [2]. In the model-based approach, we may formulate the problem as a partially observable Markov decision process (POMDP). Approximate algorithms have made dramatic progress on solving POMDPs [3], [4], [5], [6], [7]; however, manually constructing POMDP models or learning them from data remains difficult [8], [9], [10]. In the model-free approach, we circumvent the difficulty of model construction by directly searching for a solution within a policy class [11], [12]. The key issue is then choosing priors that allow efficient policy search.

Deep neural networks (DNNs) have brought unprecedented success in many domains [13], [14], [1]. Priors on the network architecture make learning efficient, e.g. convolutions allow for local, spatially invariant features [15]. DNNs provide a distinct new approach to partially observable decision-making [16], [17], [18], [19]. In particular, DQN, a convolutional architecture, has successfully tackled Atari games with complex visual input [14]. Combining DQN with recurrent LSTM layers allows them to deal with partial observability [18], [19]. However, such architectures are fundamentally limited, because their priors fail to exploit the underlying sequential nature of planning.


Fig. 1: Integrating planning and learning through algorithmic priors. The policy is represented by a model connected to an algorithm that solves the model. Both the model and the algorithm are encoded in a single neural network.

We want to combine the strength of algorithmic planning and deep learning in order to scale to the challenges of real-world decision-making. The question is then: how do we integrate the structure of algorithmic planning into a deep learning framework, i.e. what are the suitable priors on a DNN for decision making under partial observability? We propose *algorithmic priors* that integrate algorithmic planning and deep learning by embedding both a model and an algorithm that solves the model, in a single DNN architecture (Fig. 1).

We apply this approach to a challenging partially observable navigation task, where a robot is placed in a previously unseen 3-D maze and must navigate to a specified goal while avoiding unforeseen obstacles. In this decision-making task the state of the robot is partially observable. The robot receives a 2-D floor map, but it does not know its initial location on the map. Instead, it must infer it from long sequences of sensor observations from an onboard monocular camera.

We introduce the Navigation Networks (NavNet), a recurrent neural network (RNN) that employs algorithmic priors for navigation under partial observability. NavNets extend our work on QMDP-nets that addressed partially observable planning in simplified domains [20]. In contrast to QMDP-nets, NavNets embed the entire solution structure of partially observable navigation: localization, planning and acting. Localization must now deal with camera images; and a reactive actor policy is now responsible for avoiding unforeseen obstacles. In preliminary experiments NavNets successful learned policies from expert demonstrations that generalize over simulated 3-D environments. Results indicate that integrating algorithmic planning and deep learning enables reasoning with partial observations in large-scale,

[1]NUS Graduate School for Integrative Sciences and Engineering
[2]School of Computing
National University of Singapore, 119077, Singapore.
{karkus, dyhsu, leews}@comp.nus.edu.sg

visual domains. In our experiments the robot is restricted to a grid, and actions are simple, discrete motion commands; however, the approach could be applied to continuous control in the future.

## II. Related Work

The idea of embedding specific computation structures in the neural network architecture has been gaining attention recently. Previous work addressed decision making in fully observable domains such as value iteration in Markov decision processes [21], searching in graphs [22], [23], [24], path integral optimal control [25] and quadratic program optimization [26], [27]. These works do not address the issue of partial observability which drastically increases the computational complexity of decision making [2]. Another group of work addressed probabilistic state estimation [28], [29], but they do not deal with decision making or planning.

Both Shankar et al. [30] and Gupta et al. [31] addressed planning under partial observability. The former focuses on learning a model rather than a policy, where the model does not generalize over environments. The latter proposes a network learning approach to robot navigation with the focus on mapping instead of partially observable planning. They address a navigation problem in unknown environments where the location of the robot is always known. In our setting the robot location is partially observable: it must be inferred from camera images and a 2-D map.

Finally, we introduced QMDP-nets [20] that use algorithmic priors for learning partially observable decision making policies in simplified domains. In this paper we extend QMDP-nets to tackle a much more complex navigation domain that requires reasoning with visual observations and accounting for unforeseen obstacles.

## III. Algorithmic priors

We propose to use planning algorithms as "priors" on a DNN architecture for efficiently learning under partial observability. We search for a policy in the form of a DNN, but impose the structure of both a model and a planning algorithm on the DNN architecture (Fig. 1). The weights of the neural network then correspond to the model parameters, which are learned. We do not want to rely on data alone for learning the computational steps of planning. Instead, we explicitly encode a planning algorithm in a network learning architecture, and thus combine the strengths of model-free policy search and model-based planning. The core idea for encoding an algorithm in a DNN is is to view neural networks as *differentiable programs*, where algorithmic operations are realized as layers of the neural network. For example, a weighted sum becomes a convolutional layer, a maximum operation becomes a max-pool layer.

We expect that embedding a model in the network allows efficient generalization over a large task space. Embedding an algorithm allows learning end-to-end and thus circumvents the difficulties of traditional model-based learning. Moreover, we may learn abstractions that compensate for the limitations of an approximate algorithm through end-to-end training [20].

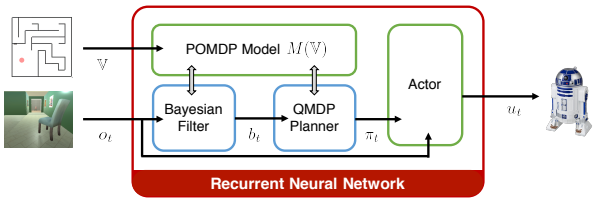


Fig. 2: NavNets are recurrent neural networks with algorithmic priors. A NavNet represents a policy, it maps from observations to actions, but it encodes the entire solution structure of navigation – state estimation, planning and acting – in a single, differentiable neural network.

Ultimately, we envision a fundamentally new approach to robotic systems design, where all building blocks are implemented as differentiable programs, and thus can be jointly optimized for the overall system objective.

## IV. Navigation networks

We define a partially observable navigation task that is prevalent in mobile robot applications. A robot is placed in a previously unseen indoor environment and must navigate to a specified goal. The robot receives a 2-D floor map that indicates walls and the goal; however, the robot does not know its own location on the map initially. Instead, it must estimate it based on past actions and observations from an onboard monocular camera. Although being uncertain of its location, the robot must choose actions that lead to the goal while avoiding walls, as well as other, previously unforeseen obstacles that were not indicated on the floor map.

We introduce the Navigation Network, a deep RNN architecture that employs algorithmic priors specific to navigation under state uncertainty (Fig. 2). Robot navigation is typically addressed by decomposing the problem to localization, planning and control. We apply the same decomposition, but encode all three components in a unified DNN representation. More specifically, the weights of a NavNet encode an abstract POMDP model, which is learned. The network also encodes an algorithm that solves the POMDP model. First, a Bayesian filter integrates information from sequences of visual observations and past actions into a belief, i.e. a probabilistic estimate of the state. Second, a QMDP planner creates a high-level plan given the map and the current belief. Finally, an actor policy takes the high-level plan and a camera observation and outputs an action. The actor generally chooses actions that execute the plan; however, it may also need to deviate from the plan to account for unforeseen obstacles blocking the path.

A navigation network encodes a POMDP model, $M(\mathbb{V}) = (S, A, O, T = f_T(\cdot|\mathbb{V}), Z = f_Z(\cdot|\mathbb{V}), R = f_R(\cdot|\mathbb{V}))$, which is explicitly conditioned on the floor map, $\mathbb{V}$. The states $s \in S$ are cells in a grid with orientation, actions $a \in A$ are discrete actions of the robot, and observations $o \in O$ are camera images. The spaces $S$, $A$ and $O$ are fixed across environments and are chosen a-priori. The rewards $R = f_R(\cdot|\mathbb{V})$, transition model $T = f_T(\cdot|\mathbb{V})$ and observation model $Z = f_Z(\cdot|\mathbb{V})$ are conditioned on the environment, i.e. the floor map $\mathbb{V}$. They

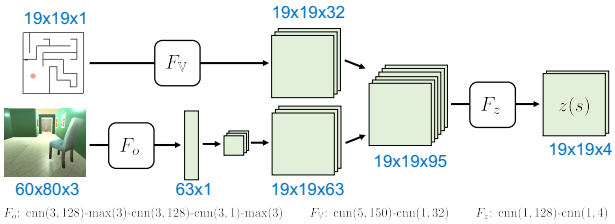


Fig. 3: Observation model that maps from a camera image and the floor map to the posterior $z(s) = p(s|o_t, \mathbb{V})$. We use $\text{cnn}(k, f)$ to denote convolution with $k \times k$ kernel and $f$ filters; and $\max(k)$ to denote max-pooling with $k \times k$ kernel.

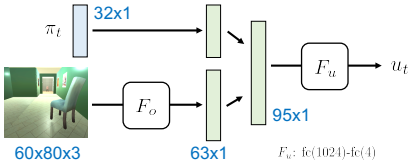


Fig. 4: The actor component maps from a camera image and a vector representation of the high-level plan, $\pi_t$, to an action, $u_t$. The $F_o$ component in Fig. 3 and Fig. 4 has the same structure but the weights are not tied. We use $\text{fc}(h)$ to denote a fully connected layer with $h$ output units.

are represented by distinct neural network blocks, and are learned through end-to-end training.

The Bayesian filter updates the belief iteratively through

$$b'_t(s) = \sum_{s' \in S} T(s', \underline{a}_t, s) b_t(s'), \tag{1}$$

$$b_{t+1}(s) = \eta Z(s, \underline{o}_t) b'_t(s), \tag{2}$$

where $b_t(s)$ is the belief over states $s \in S$ at time $t$; $o_t \in O$ is the observation received after taking action $a_t \in A$; and $\eta$ is a normalization factor. The QMDP algorithm approximates the solution to the planning problem through dynamic programming. It first takes $K$ steps of value iteration,

$$Q_{k+1}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_k(s'), \tag{3}$$

$$V_k(s) = \max_a Q_k(s, a). \tag{4}$$

where $k = 1..K$ is the planning step, $V$ and $Q$ are state and state-action values, respectively. The algorithm then chooses an action based on an approximation of its future value,

$$q(a) = \sum_{s \in S} Q_K(s, a) b_t(s). \tag{5}$$

We embed the filter and the planner in a single neural network (Fig. 2) by implementing both as differentiable programs, i.e. we express the algorithmic operations (1 – 5) as convolutional, linear and max-pool layers. The operations are applied to the components of the POMDP model, $f_T$, $f_Z$ and $f_R$, all of which are represented by neural networks and are learned. We do not use supervision on the model components. Instead, we expect a useful model to emerge through training the policy end-to-end.

The neural network architecture is similar to QMDP-nets [20], with the notable exception of the observation model and the reactive actor component. In our experiments the robot is restricted to a discrete grid of size $M \times N$ and has $L$ possible orientations. The input map is a $M \times N$ image. The belief is then represented by a $M \times N \times L$ tensor. $f_T(\cdot)$ is a single $3 \times 3$ convolutional layer with $L \cdot |A|$ output channels, one for each discrete orientation and action pair. $f_R(\cdot|\mathbb{V})$ is a two-layer CNN, with $3 \times 3$ and $1 \times 1$ kernels, and 150 convolutional filters. The input is the map $\mathbb{V}$; the output is a $M \times N \times L \cdot |A|$ tensor corresponding to rewards in each state-action pair.

The observation model, $f_Z$, is the most complex component of the learned POMDP model. Unlike in QMDP-nets, we must now deal with visual observations. This involves inferring the environment geometry from a camera image and matching it against the 2-D floor map. Learning the joint probability distribution $Z(s, o|\mathbb{V})$ for the large space of image observations is intractable. Instead, we directly learn the posterior $z(s) = p(s|o_t, \mathbb{V})$ and plug it in to (2). The implementation of this component is show in Fig. 3.

Navigation networks also include a low-level actor policy that executes the high-level plan while avoiding obstacles that the plan could not account for. The actor takes in a camera observation, $o_t$, and a vector representation of the high-level plan, $\pi_t$; and outputs a low-level action, $u_i$. In our experiments $u$ and $a$ are defined in the same discrete space corresponding to actions in a discrete grid; but $u$ could be continuous velocity or torque signal in future work. The high-level plan is represented by a vector of the computed $Q$ values for each high-level action, i.e. $\pi_t = q(a)$ for $a \in A$. In addition, we found that a few steps of history helps the actor to avoid certain oscillating behaviors. Therefore, we include in $\pi_t$ four steps of past weighted Q values, $\pi_{t-\tau}$, and four steps of previous action outputs, $u_{t-1-\tau}$ where $\tau = 1..4$. The network architecture is shown in Fig. 4.

## V. Preliminary Experiments

We evaluated the navigation networks for variants of the partially observable navigation task in a custom-built, high-fidelity simulator based on the Unity 3D Engine [32]. The robot is placed in a randomly generated 3-D environment, where it is restricted to a $19 \times 19$ discrete grid and 4 possible orientations. The action are simple motion commands: move forward, turn left, turn right, and stay put. The robot does not know its own state initially. Instead, it receives a 2-D map of the environment ($19 \times 19$ binary image) and must estimate its location based on camera observations ($60 \times 80$ RGB images) rendered from the 3-D scene. We place additional objects in the environment at random locations, without fully blocking a passage. The objects are picked randomly from a set of 23 common household furniture such as chairs, tables, beds, etc. Examples are shown in Fig. 5. We evaluate collisions assuming the robot occupies the entire grid cell it is located in.

We define three variants of the navigation task as follows:

Fig. 5: Examples for floor maps and camera observations from the training set (top) and test set (bottom). The environment layout and the textures are randomly generated.

- **Task A**. In this setting objects (apart from walls) act solely as visual obstruction: the robot can go through them, and they are not shown on the floor map. With this tasks we evaluate the ability of localizing given camera images and a floor map, and navigating to distant goals in a new environment.
- **Task B**. The robot can no longer move through objects, but the robot's map – unlike a typical real-life floor map – includes all objects in the environment such as chairs, tables, etc. The robot must recognize objects and their relative position from camera images, and account for them for both localization and planning.
- **Task C**. In this most difficult setting the robot must avoid all objects in the environment, but its floor map only indicates the walls. Planning is unaware of objects not shown on the map, therefore the actor may need to deviate locally from the high-level plan in order to avoid unforeseen obstacles.

We trained navigation networks from expert demonstrations in a set of $10k$ environments. We then evaluated the learned policies in a separate set of 200 random environments. For training we used successful trajectories produced by a clairvoyant QMDP policy, 5 trajectories for each environment. The clairvoyant expert has access to all objects on the map and receives binary observations that define the occupancy of grid cells in front of the robot.

The training was carried out by backpropagation through time using a cross-entropy loss between demonstrated and predicted action outputs, thus learning a policy end-to-end without supervision on the underlying POMDP model. We used multiple steps of curriculum. First, we trained a policy in synthetic grids that share the underlying structure of planning, but only involves simple binary observations. We then trained further for Task A, Task B and Task C in a sequence, gradually increasing the difficulty[1]. We used the full network architecture described in the previous section for Task C, while for Task A and Task B we replaced the actor component by a single fully connected layer.

[1]For Task C we initialized the planner and filter weights from Task A, while the weights of the visual processing block in the actor component were initialized from Task B.

TABLE I: Summary of results comparing navigation network policies (NavNet) and a clairvoyant QMDP policy (QMDP).

| Objects on floor map<br>Collider objects | **Task A**<br>walls only<br>walls only | **Task B**<br>all objects<br>all objects | **Task C**<br>walls only<br>all objects |
|---|---|---|---|
| NavNet success rate<br>(excluding collisions) | **97.5%**<br>(97.0%) | **96.5%**<br>(95.5%) | **91.5%**<br>(83.0%) |
| NavNet collision rate | 1.5% | 1.5% | 14.5% |
| NavNet time | 37.7 | 34.4 | 36.5 |
| QMDP success rate | 81.1% | 84.1% | 62.6% |
| QMDP collision rate | 0.0% | 0.0% | 0.0% |
| QMDP steps | 32.8 | 31.1 | 32.1 |

## VI. Results

Preliminary results are summarized in Table I. We report success rate (trials that reached the goal); success rate excluding collisions (collision free trials that reached the goal); collision rate (trials that involved one or more collisions); and average time (steps required to reach the goal, averaged for successful trials only).

Learned policies were able to successfully reach the goal in a reasonable number of steps with no collisions for the majority of the environments.

We compared to a clairvoyant QMDP policy that has access to much simpler binary observations and a map with all obstacles; and it plans with the "true" underlying POMDP model. The true POMDP model would give a perfect solution to our learning problem if the planning algorithm is exact; but not necessarily if the algorithm is approximate. An "incorrect", but useful model may compensate the limitations of an approximation algorithm, in a way similar to reward shaping in reinforcement learning [33]. In our experiments learned NavNets performed significantly better than the clairvoyant QMDP. The QMDP algorithm is the same for both the clairvoyant QMDP and the NavNets, but end-to-end training allowed learning a model that is more effective for the approximate QMDP algorithm. We note that the clairvoyant QMDP was used to generate the expert data for training, but we excluded unsuccessful demonstrations. When including both successful and unsuccessful demonstrations NavNets did not perform better than QMDP, as expected.

The algorithmic priors on the DNN architecture enabled policies to generalize efficiently to previously unseen 3-D environments, by carrying over the shared structure of the underlying reasoning required for planning. While we defer direct comparison with alternative DNN architectures to future work, we note that in previous reports DNNs without algorithmic priors were unable to learn navigation policies even in much simpler settings [20].

While initial results are promising, the success rate and collision rate in the more difficult setting (Task C) are not yet satisfactory for a real-world application. We observed that many failures are caused by the inability of avoiding obstacles that were not anticipated by the plan. A possible reason for this is that without memory, the actor can only

alter the plan when an obstacle is visible. It also cannot choose good alternative path locally, because it is unaware of the goal and the plan apart from the value of the immediate next step. We may address these deficiencies by feeding in multiple steps of past camera observations to the actor, as well as a larger local "section" of the high-level plan.

## VII. Discussion

We proposed algorithmic priors to integrate planning with deep learning. In this section we discuss when algorithmic priors can be expected to be effective; and identify key challenges for future research.

When comparing to standard model-free learning we should consider the following. Certain tasks are hard because they require a complex model to describe, however, solving the model can yield a simple policy. Other tasks require a complex policy, but the policy can be derived from a moderately complex model through tractable planning. Model-free policy learning can be effective in the former case, while we expect algorithmic priors to be critically important in the latter case. We can apply the same reasoning when decomposing a complex problem to sub-problems: we can employ different degrees of algorithmic priors depending on the nature of a sub-task. For example, in navigation networks the filter and planner components encode strong algorithmic priors for dealing with partial observability, but we do not employ algorithmic priors for low-level actor, because this sub-task is expected to be reactive in nature.

How does the approach compare to traditional model-based learning? Model learning is often hard because of the difficulty of inferring model parameters from the available data; or because small model errors are amplified through planning. Embedding the model and the algorithm in the same neural network allows learning end-to-end, which may in turn circumvent the difficulties of conventional model learning.

In order to embed an algorithm in a neural network, we must implement it as a differentiable program. However, some algorithmic operations are not differentiable, such as indexing, sampling or argmax. We may deal with such operations by developing their differentiable approximations, e.g. soft-indexing in QMDP-nets [20]; or by analytically approximating gradients of larger computational blocks [26]. Another concern is that repeated computation, as well as more sophisticated algorithms, render large neural networks, that are in turn hard to train. Differential programs are certainly limited in terms of algorithmic computation, but they allow learning abstract models or a suitable search space end-to-end, and thus may reduce the required planning computation significantly. Results on navigation networks and QMDP-nets demonstrate that this is indeed possible; however, we believe that learning more aggressive abstractions will be important in scaling to more difficult tasks.

In partially observable domains, a particularly important issue is the representation of beliefs, i.e. probability distributions over states. Modern POMDP algorithms make planning tractable by sampling from the belief and reasoning with particles [6], [7]. An exciting line of future work may explore encoding more scalable belief representations, such as particle beliefs, in a differentiable neural network.

## VIII. Conclusion

We proposed to integrate planning and learning through algorithmic priors. Algorithmic priors impose the structure of planning on a DNN architecture, by viewing DNNs as differentiable programs instead of parametric function approximators. Implementing all components of a larger robotic system by differentiable programs would allow jointly optimizing the entire system for a given task. In this paper we made a step towards this vision by encoding state estimation, planning and acting in a single neural network for a challenging partially observable navigation task.

There are several exciting directions for future work that deal with high-dimensional state spaces for planning; encode more capable algorithms in the neural network; or explore how differentiable modules can be effectively pre-trained and transferred over domains.

## References


[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[3] J. Pineau, G. J. Gordon, and S. Thrun, "Applying metric-trees to belief-point POMDPs," in *Advances in Neural Information Processing Systems*, 2003, p. None.

[4] M. T. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[5] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based POMDP planning by approximating optimally reachable belief spaces." in *Robotics: Science and Systems*, 2008.

[6] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Advances in Neural Information Processing Systems*, 2010, pp. 2164–2172.

[7] N. Ye, A. Somani, D. Hsu, and W. S. Lee, "Despot: Online POMDP planning with regularization," *Journal of Artificial Intelligence Research*, vol. 58, pp. 231–266, 2017.

[8] M. L. Littman, R. S. Sutton, and S. Singh, "Predictive representations of state," in *Advances in Neural Information Processing Systems*, 2002, pp. 1555–1562.

[9] G. Shani, R. I. Brafman, and S. E. Shimony, "Model-based online learning of POMDPs," in *European Conference on Machine Learning*, 2005, pp. 353–364.

[10] B. Boots, S. M. Siddiqi, and G. J. Gordon, "Closing the learning-planning loop with predictive state representations," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 954–966, 2011.

[11] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.

[12] J. A. Bagnell, S. Kakade, A. Y. Ng, and J. G. Schneider, "Policy search by dynamic programming," in *Advances in Neural Information Processing Systems*, 2003, pp. 831–838.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] B. Bakker, V. Zhumatiy, G. Gruener, and J. Schmidhuber, "A robot that reinforcement-learns to identify and memorize important previous observations," in *International Conference on Intelligent Robots and Systems*, 2003, pp. 430–435.

[18] M. J. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," *arXiv preprint, arXiv:1507.06527*, 2015.

[19] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[20] P. Karkus, D. Hsu, and W. S. Lee, "QMDP-net: Deep learning for planning under partial observability," in *Advances in Neural Information Processing Systems*, 2017, pp. 4697–4707.

[21] A. Tamar, S. Levine, P. Abbeel, Y. Wu, and G. Thomas, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2146–2154.

[22] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *Advances in Neural Information Processing Systems*, 2017, pp. 6120–6130.

[23] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson, "TreeQN and ATreeC: Differentiable tree planning for deep reinforcement learning," *arXiv preprint, arXiv:1710.11417*, 2017.

[24] A. Guez, T. Weber, I. Antonoglou, K. Simonyan, O. Vinyals, D. Wierstra, R. Munos, and D. Silver, "Learning to search with MCTSnets," *arXiv preprint, arXiv:1802.04697*, 2018.

[25] M. Okada, L. Rigazio, and T. Aoshima, "Path integral networks: End-to-end differentiable optimal control," *arXiv preprint arXiv:1706.09597*, 2017.

[26] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, 2017, pp. 136–145.

[27] P. Donti, B. Amos, and J. Z. Kolter, "Task-based end-to-end model learning in stochastic optimization," in *Advances in Neural Information Processing Systems*, 2017, pp. 5484–5494.

[28] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel, "Backprop KF: Learning discriminative deterministic state estimators," in *Advances in Neural Information Processing Systems*, 2016, pp. 4376–4384.

[29] R. Jonschkowski and O. Brock, "End-to-end learnable histogram filters," in *Workshop on Deep Learning for Action and Interaction at NIPS*, 2016. [Online]. Available: http://www.robotics.tu-berlin.de/fileadmin/fg170/Publikationen_pdf/Jonschkowski-16-NIPS-WS.pdf

[30] T. Shankar, S. K. Dwivedy, and P. Guha, "Reinforcement learning via recurrent convolutional neural networks," in *International Conference on Pattern Recognition*, 2016, pp. 2592–2597.

[31] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," *arXiv preprint arXiv:1702.03920*, 2017.

[32] "Unity 3D game engine." [Online]. Available: http://unity3d.com

[33] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning*, 1999, pp. 278–287.