Long-term Planning by Short-term Prediction

Shai Shalev-Shwartz Nir Ben-Zrihem Aviad Cohen Amnon Shashua

Mobileye

Abstract

We consider planning problems, that often arise in autonomous driving applications, in which an agent should decide on immediate actions so as to optimize a long term objective. For example, when a car tries to merge in a roundabout it should decide on an immediate acceleration/braking command, while the long term effect of the command is the success/failure of the merge. We propose to tackle the planning problem by decomposing the problem into two phases: First, we apply supervised learning for predicting the near future based on the present. We require that the predictor will be differentiable with respect to the representation of the present. Second, we model a full trajectory of the agent using a recurrent neural network, where unexplained factors are modeled as (additive) input nodes. This allows us to solve the long-term planning problem using supervised learning techniques and direct optimization over the recurrent neural network. Our approach enables us to learn robust policies by incorporating adversarial elements to the environment.

1 Introduction

Two of the most crucial elements of autonomous driving systems are sensing and planning. Sensing deals with finding a compact representation of the present state of the environment, while planning deals with deciding on what actions to take so as to optimize future objectives. Supervised machine learning techniques are very useful for solving sensing problems. In this paper we describe a machine learning algorithmic framework for the planning part. Traditionally, machine learning approaches for planning are studied under the framework of Reinforcement Learning (RL) — see [4, 10, 22, 23] for a general overview and [12] for a comprehensive review of reinforcement learning in robotics.

Typically, RL is performed in a sequence of consecutive rounds. At round t, the planner (a.k.a. the agent) observes a state, $s_t \in S$, which represents the agent as well as the environment. It then should decide on an action $a_t \in A$. After performing the action, the agent receives an immediate reward, $r_t \in \mathbb{R}$, and is moved to a new state, s_{t+1}. As a simple example, consider an adaptive cruise control (ACC) system, in which a self driving vehicle should implement acceleration/braking so as to keep an adequate distance to a preceding vehicle while maintaining smooth driving. We can model the state as a pair, $s_t = (x_t, v_t) \in \mathbb{R}^2$, where x_t is the distance to the preceding vehicle and v_t is the velocity of the car relative to the velocity of the preceding vehicle. The action $a_t \in \mathbb{R}$ will be the acceleration command (where the car slows down if $a_t < 0$). The reward can be some function that depends on $|a_t|$ (reflecting the smoothness of driving) and on s_t (reflecting that we keep a safe distance from the preceding vehicle). The goal of the planner is to maximize the cumulative reward (maybe up to a time horizon or a discounted sum of future rewards). To do so, the planner relies on a policy, $\pi : S \to A$, which maps a state into an action.

Supervised Learning (SL) can be viewed as a special case of RL, in which s_t is sampled i.i.d. from some distribution over S and the reward function has the form $r_t = -\ell(a_t, y_t)$, where ℓ is some loss function, and the learner observes the value of y_t which is the (possibly noisy) value of the optimal action to take when viewing the state s_t.

There are several key differences between the fully general RL model and the specific case of SL. These differences makes the general RL problem much harder.

1. In SL, the actions (or predictions) taken by the learner have no effect on the environment. In particular, s_{t+1} and a_t are independent. This has two important implications:
   - In SL we can collect a sample $(s_1, y_1), \ldots, (s_m, y_m)$ in advance, and only then search for a policy (or predictor) that will have good accuracy on the sample. In contrast, in RL, the state s_{t+1} usually depends

on the action (and also on the previous state), which in turn depends on the policy used to generate the action. This ties the data generation process to the policy learning process.

- Because actions do not effect the environment in SL, the contribution of the choice of a_t to the performance of π is local, namely, a_t only affects the value of the immediate reward. In contrast, in RL, actions that are taken at round t might have a long-term effect on the reward values in future rounds.

2. In SL, the knowledge of the "correct" answer, y_t, together with the shape of the reward, $r_t = -\ell(a_t, y_t)$, gives us a full knowledge of the reward for all possible choices of a_t. Furthermore, this often enables us to calculate the derivative of the reward with respect to a_t. In contrast, in RL, we only observe a "one-shot" value of the reward for the specific choice of action we took. This is often called a "bandit" feedback. It is one of the main reasons for the need of "exploration", because if we only get to see a "bandit" feedback, we do not always know if the action we took is the best one.

Before explaining our approach for tackling these difficulties, we briefly describe the key idea behind most common reinforcement learning algorithms. Most of the algorithms rely in some way or another on the mathematically elegant model of a Markov Decision Process (MDP), pioneered by the work of Bellman [2, 3]. The Markovian assumption is that the distribution of s_{t+1} is fully determined given s_t and a_t. This yields a closed form expression for the cumulative reward of a given policy in terms of the stationary distribution over states of the MDP. The stationary distribution of a policy can be expressed as a solution to a linear programming problem. This yields two families of algorithms: optimizing with respect to the primal problem, which is called policy search, and optimizing with respect to the dual problem, whose variables are called the *value function*, V^π. The value function determines the expected cumulative reward if we start the MDP from the initial state s, and from there on pick actions according to π. A related quantity is the state-action value function, $Q^\pi(s, a)$, which determines the cumulative reward if we start from state s, immediately pick action a, and from there on pick actions according to π. The Q function gives rise to a crisp characterization of the optimal policy (using the so called Bellman's equation), and in particular it shows that the optimal policy is a deterministic function from S to A (in fact, it is the greedy policy with respect to the optimal Q function).

In a sense, the key advantage of the MDP model is that it allows us to couple all the future into the present using the Q function. That is, given that we are now in state s, the value of $Q^\pi(s, a)$ tells us the effect of performing action a at the moment on the entire future. Therefore, the Q function gives us a local measure of the quality of an action a, thus making the RL problem more similar to SL.

Most reinforcement learning algorithms approximate the V function or the Q function in one way or another. Value iteration algorithms, e.g. the Q learning algorithm [26], relies on the fact that the V and Q functions of the optimal policy are fixed points of some operators derived from Bellman's equation. Actor-critic policy iteration algorithms aim to learn a policy in an iterative way, where at iteration t, the "critic" estimates Q^{π_t} and based on this, the "actor" improves the policy.

Despite the mathematical elegancy of MDPs and the conveniency of switching to the Q function representation, there are several limitations of this approach. First, as noted in [12], usually in robotics, we may only be able to find some approximate notion of a Markovian behaving state. Furthermore, the transition of states depends not only on the agent's action, but also on actions of other players in the environment. For example, in the ACC example mentioned previously, while the dynamic of the autonomous vehicle is clearly Markovian, the next state depends on the behavior of the other driver, which is not necessarily Markovian. One possible solution to this problem is to use partially observed MDPs [27], in which we still assume that there is a Markovian state, but we only get to see an observation that is distributed according to the hidden state. A more direct approach considers game theoretical generalizations of MDPs, for example the Stochastic Games framework. Indeed, some of the algorithms for MDPs were generalized to multi-agents games. For example, the minimax-Q learning [14] or the Nash-Q learning [9]. Other approaches to Stochastic Games are explicit modeling of the other players, that goes back to Brown's fictitious play [6], and vanishing regret learning algorithms [8, 7]. See also [25, 24, 11, 5]. As noted in [20], learning in multi-agent setting is inherently more complex than in the single agent setting.

A second limitation of the Q function representation arises when we depart from a tabular setting. The tabular setting is when the number of states and actions is small, and therefore we can express Q as a table with $|S|$ rows and $|A|$ columns. However, if the natural representation of S and A is as Euclidean spaces, and we try to discretize the state

and action spaces, we obtain that the number of states/actions is exponential in the dimension. In such cases, it is not practical to employ the tabular setting. Instead, the Q function is approximated by some function from a parametric hypothesis class (e.g. neural networks of a certain architecture). For example, the deep-Q-networks (DQN) learning algorithm of [16] has been successful at playing Atari games. In DQN, the state space can be continuous but the action space is still a small discrete set. There are approaches for dealing with continuous action spaces (e.g. [21]), but they again rely on approximating the Q function. In any case, the Q function is usually very complicated and sensitive to noise, and it is therefore quite hard to learn it. Indeed, it was observed that value based methods rarely work out-of-the-box in robotic applications [12], and that the best performing methods rely on a lot of prior knowledge and reward shaping [13, 17]. Intuitively, the difficulty in learning Q is that we need to implicitly understand the dynamics of the underlying Markov process.

A radically different approach has been introduced by Schmidhuber [19], who tackled the RL problem using a recurrent neural network (RNN). Following [19], there have been several additional algorithms that rely on RNNs for RL problems. For example, Backer [1] proposed to tackle the RL problem using recurrent networks with the LSTM architecture. His approach still relies on the value function. Schäfer [18] used RNN to model the dynamics of partially observed MDPs. Again, he still relies on explicitly modeling the Markovian structure. There have been few other approaches to tackle the RL problem without relying on value functions. Most notably is the REINFORCE framework of Williams [28]. It has been recently successful for visual attention [15, 29]. As already noted by [19], the ability of REINFORCE to estimate the derivative of stochastic units can be straightforwardly combined within the RNN framework.

In this paper we combine Schmidhuber's approach, of tackling the policy learning problem directly using a RNN, with the notions of multi-agents games and robustness to adversarial environments from the game theory literature. Furthermore, we do not explicitly rely on any Markovian assumption. Our approach is described in the next section.

2 Planning by Prediction

Throughout, we assume that the state space, S, is some subset of $\mathbb{R}^d$, and the action space, A, is some subset of $\mathbb{R}^k$. This is the most natural representation in many applications, and in particular, the ones we describe in Section 3.

Recall that we discussed two key differences between RL and SL: (1) Since the past actions affect future rewards, we must propagate information from the future back to the past. (2) The "bandit" nature of the rewards blurs the dependence between (state,action) and reward, which complicates the learning process.

We first make the trivial observation that there are interesting problems in which the second difference is not an issue. For example, in Section 3 we define a reward value for the ACC application which is differentiable with respect to the current state and action. In fact, even if the reward is given in a "bandit" manner, the problem of learning a differentiable function $\hat{r}(s,a)$ such that $\hat{r}(s_t,a_t) \approx r_t$ is a relatively straightforward SL problem — it is a one dimensional regression problem. Therefore, the first step of our approach is to either define the reward as a function $\hat{r}(s,a)$, which is differentiable with respect to s and a, or to use a regression learning algorithm in order to learn a differentiable function $\hat{r}$ that minimizes some regression loss over a sample with instance vector being $(s_t,a_t) \in \mathbb{R}^d \times \mathbb{R}^k$ and target scalar being r_t. In some situations, to create the training set we need to use some elements of exploration. Since this is a standard technique, we omit the details.

To tackle the connection between past and future we use a similar idea: suppose we can learn a differentiable function $\hat{N}(s,a)$ such that $\hat{N}(s_t,a_t) \approx s_{t+1}$. Learning such a function is again a SL problem. We can think about $\hat{N}$ as a predictor for the near future. Next, the policy that maps from S to A will be described using a parametric function $\pi_\theta : S \to A$. If we express π_θ as a neural network, we can express an episode of running the agent for T rounds using a recurrent neural network (RNN), where the next state is defined as $s_{t+1} = \hat{N}(s_t,a_t) + \nu_t$. Here, $\nu_t \in \mathbb{R}^d$ is defined by the environment, and expresses the unpredictable aspects of the near future. The fact that s_{t+1} depends on s_t and a_t in a differentiable manner enables us to connect between future reward values and past actions. We can therefore learn the parameter vector of the policy function, π_θ, by back-propagation over the resulted RNN.

Note that we do not impose explicit probabilistic assumptions on ν_t. In particular, we do not require Markovian relation. Instead, we rely on the recurrent network to propagate "enough" information between past and future. Intuitively, $\hat{N}(s_t,a_t)$ describes the predictable part of the near future, while ν_t expresses the unpredictable aspects, usually due to the behavior of other players in the environment. The learner should learn a policy that will be robust to the

behavior of other players. Naturally, if $\|\nu_t\|$ is large, the connection between our past actions and future reward values will be too noisy for learning a meaningful policy.

As noted in [19], explicitly expressing the dynamic of the system in a transparent way enables to incorporate prior knowledge more easily. For example, in Section 3 we demonstrate how prior knowledge greatly simplifies the problem of defining $\hat{N}$.

2.1 Robustness to Adversarial Environment

Since our model does not impose probabilistic assumptions on ν_t, we can consider environments in which ν_t is being chosen in an adversarial manner. Of course, we must make some restrictions on μ_t, otherwise the adversary can make the planning problem impossible. A natural restriction is to require that $\|\mu_t\|$ is bounded by a constant. Robustness against adversarial environment is quite useful in autonomous driving applications. We describe a real world aspect of adversarial environment in Section 3.

Here, we show that choosing μ_t in an adversarial way might even speed up the learning process, as it can focus the learner toward the robust optimal policy. We consider the following simple game. The state is $s_t \in \mathbb{R}$, the action is $a_t \in \mathbb{R}$, and the immediate loss function is $0.1|a_t| + [|s_t| - 2]_+$, where $[x]_+ = \max\{x, 0\}$ is the ReLU function. The next state is $s_{t+1} = s_t + a_t + \nu_t$, where $\nu_t \in [-0.5, 0.5]$ is chosen by the environment in an adversarial manner.

It is possible to see that the optimal policy can be written as a two layer network with ReLU: $a_t = -[s_t - 1.5]_+ + [-s_t - 1.5]_+$. Observe that when $|s_t| \in (1.5, 2]$, the optimal action has a larger immediate loss than the action $a = 0$. Therefore, the learner must plan for the future and cannot rely solely on the immediate loss.

Observe that the derivative of the loss w.r.t. a_t is $0.1\,\mathrm{sign}(a_t)$ and the derivative w.r.t. s_t is $1[|s_t| > 2]\,\mathrm{sign}(s_t)$. Suppose we are in a situation in which $s_t \in (1.5, 2]$. The adversarial choice of ν_t would be to set $\nu_t = 0.5$, and therefore, we will have a non-zero loss on round $t + 1$, whenever $a_t > 1.5 - s_t$. In all such cases, the derivative of the loss will back-propagate directly to a_t. We therefore see that the adversarial choice of ν_t helps the learner to get a non-zero back-propagation message in all cases for which the choice of a_t is sub-optimal.

3 Example Applications

The goal of this section is to demonstrate some aspects of our approach on two toy examples: adaptive cruise control (ACC) and merging into a roundabout.

3.1 The ACC Problem

In the ACC problem, a host vehicle is trying to keep an adequate distance of 1.5 seconds to a target car, while driving as smooth as possible. We provide a simple model for this problem as follows. The state space is $\mathbb{R}^3$ and the action space is $\mathbb{R}$. The first coordinate of the state is the speed of the target car, the second coordinate is the speed of the host car, and the last coordinate is the distance between host and target (namely, location of the host minus location of the target on the road curve). The action to be taken by the host is the acceleration, and is denoted by a_t. We denote by τ the difference in time between consecutive rounds (in the experiment we set τ to be 0.1 seconds).

Denote $s_t = (v_t^{\text{target}}, v_t^{\text{host}}, x_t)$ and denote by a_t^{target} the (unknown) acceleration of the target. The full dynamics of the system can be described by:

$$\begin{aligned} v_t^{\text{target}} &= [v_{t-1}^{\text{target}} + \tau\, a_{t-1}^{\text{target}}]_+ \\ v_t^{\text{host}} &= [v_{t-1}^{\text{host}} + \tau\, a_{t-1}]_+ \\ x_t &= [x_{t-1} + \tau\,(v_{t-1}^{\text{target}} - v_{t-1}^{\text{host}})]_+ \end{aligned}$$

This can be described as a sum of two vectors:

$$\begin{aligned} s_t &= ([s_{t-1}[0] + \tau a_{t-1}^{\text{target}}]_+, [s_{t-1}[1] + \tau a_{t-1}]_+, [s_{t-1}[2] + \tau(s_{t-1}[0] - s_{t-1}[1])]_+) \\ &= \underbrace{(s_{t-1}[0], [s_{t-1}[1] + \tau a_{t-1}]_+, [s_{t-1}[2] + \tau(s_{t-1}[0] - s_{t-1}[1])]_+)}_{\hat{N}(s_{t-1}, a_t)} + \underbrace{([s_{t-1}[0] + \tau a_{t-1}^{\text{target}}]_+ - s_{t-1}[0], 0, 0)}_{\nu_t} \end{aligned}$$

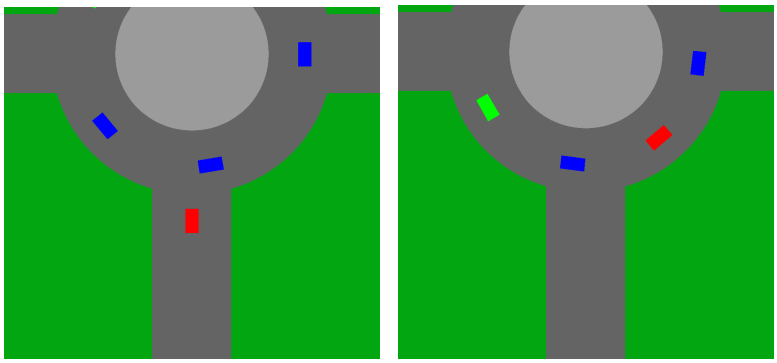

Figure 1: Screenshots from the game. The agent is the car in red. Target vehicles are in blue (aggressive cars) and in green (defensive cars). The agent doesn't observe the type of the target cars. It should infer it from their position and acceleration. On the left, the agent correctly understands that the approaching car is aggressive and therefore stop and wait. On the right we see a successful merge.

The first vector is the predictable part and the second vector is the unpredictable part.

The reward on round t is defined as follows:

$$-r_t \;=\; 0.1\,|a_t| \;+\; [|x_t/x_t^* - 1| - 0.3]_+ \quad \text{where} \quad x_t^* = \max\{1, 1.5\,v_t^{\text{host}}\}$$

The first term above penalizes for any non-zero acceleration, thus encourages smooth driving. The second term depends on the ratio between the distance to the target car, x_t, and the desired distance, x_t^*, which is defined as the maximum between a distance of 1 meter and brake distance of 1.5 seconds. Ideally, we would like this ratio to be exactly 1, but as long as this ratio is in $[0.7, 1.3]$ we do not penalize the policy, thus allowing the car some slack (which is important for maintaining a smooth drive).

3.2 Merging into a Roundabout

In this experiment, the goal of the agent is to pass a roundabout. An episode starts when the agent is approaching the bottom entrance of the roundabout. The episode ends when the agent reaches the second exit of the roundabout, or after a fixed number of steps. A successful episode is measured first by keeping a safety distance from all other vehicles in the roundabout at all times. Second, the agent should finish the route as quickly as possible. And third, it should adhere a smooth acceleration policy. At the beginning of the episode, we randomly place N_T target vehicles on the roundabout.

To model a blend of adversarial and typical behavior, with probability p, a target vehicle is modeled by an "aggressive" driving policy, that accelerates when the host tries to merge in front of it. With probability $1 - p$, the target vehicle is modeled by a "defensive" driving policy that deaccelerate and let the host merge in. In our experiments we set $p = 0.5$. The agent has no information about the type of the other drivers. These are chosen at random at the beginning of the episode.

We represent the state as the velocity and location of the host (the agent), and the locations, velocities, and accelerations of the target vehicles. Maintaining target accelerations is vital in order to differentiate between aggressive and defensive drivers based on the current state. All target vehicles move on a one-dimensional curve that outlines the roundabout path. The host vehicle moves on its own one-dimensional curve, which intersects the targets' curve at the merging point, and this point is the origin of both curves. To model reasonable driving, the absolute value of all vehicles' accelerations are upper bounded by a constant. Velocities are also passed through a ReLU because driving backward is not allowed. Note that by not allowing driving backward we make long-term planning a necessity (the agent cannot regret on its past action).

Recall that we decompose the next state, s_{t+1}, into a sum of a predictable part, $\hat{N}(s_t, a_t)$, and a non-predictable part, ν_t. In our first experiment, we let $\hat{N}(s_t, a_t)$ be the dynamics of locations and velocities of all vehicles (which

are well defined in a differentiable manner), while ν_t is the targets' acceleration. It is easy to verify that $\hat{N}(s_t, a_t)$ can be expressed as a combination of ReLU functions over an affine transformation, hence it is differentiable with respect to s_t and a_t. The vector ν_t is defined by a simulator in a non-differentiable manner, and in particular implement aggressive behavior for some targets and defensive behavior for other targets. Two frames from the simulator are showin in Figure 1. As can be seen in the supplementary videos[1], the agent learns to slowdown as it approaches the entrance of the roundabout. It also perfectly learned to give way to aggressive drivers, and to safely continue when merging in front of defensive ones.

Our second experiment is more ambitious: we do not tell the network the function $\hat{N}(s_t, a_t)$. Instead, we express $\hat{N}$ as another learnable part of our recurrent network. Besides the rewards for the policy part, we add a loss term of the form $\|\hat{N}(s_t, a_t) - s_{t+1}\|^2$, where s_{t+1} is the actual next state as obtained by the simulator. That is, we learn the prediction of the near future, $\hat{N}$, and the policy that plan for the long term, π_θ, concurrently. While this learning task is more challenging, as can be seen in the supplementary videos, the learning process still succeeds.

4 Discussion

We have presented an approach for learning driving polices in the presence of other adversarial cars using recurrent neural networks. Our approach relies on partitioning of the near future into a predictable part and an un-predictable part. We demonstrated the effectiveness of the learning procedure for two simple tasks: adaptive cruise control and roundabout merging. The described technique can be adapted to learning driving policies in other scenarios, such as lane change decisions, highway exit and merge, negotiation of the right of way in junctions, yielding for pedestrians, as well as complicated planning in urban scenarios.

References

[1] Bram Bakker. Reinforcement learning with long short-term memory. In *NIPS*, pages 1475–1482, 2001.

[2] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

[3] Richard Bellman. *Introduction to the mathematical theory of control processes*, volume 2. IMA, 1971.

[4] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[5] Ronen I Brafman and Moshe Tennenholtz. R-max–a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[6] George W Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

[7] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[8] S. HART and A. MAS-COLELL. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5), 2000.

[9] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *The Journal of Machine Learning Research*, 4:1039–1069, 2003.

[10] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285, 1996.

[11] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[1] `http://www.mobileye.com/mobileye-research/long-term-planning-by-short-term-prediction/`

[12] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, page 0278364913495721, 2013.

[13] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[14] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163, 1994.

[15] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[17] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[18] Anton Maximilian Schäfer. *Reinforcement Learning with Recurrent Neural Network*. PhD thesis, Universitat Osnabruck, 2008.

[19] Jürgen Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In *NIPS*, 1991.

[20] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007.

[21] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[23] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010. URL `http://www.ualberta.ca/~szepesva/RLBook.html`.

[24] S. Thrun. Learning to play the game of chess. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems (NIPS) 7*, Cambridge, MA, 1995. MIT Press.

[25] William Uther and Manuela Veloso. Adversarial reinforcement learning. Technical report, Technical report, Carnegie Mellon University, 1997. Unpublished, 1997.

[26] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[27] Chelsea C White III. A survey of solution techniques for the partially observed markov decision process. *Annals of Operations Research*, 32(1):215–230, 1991.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[29] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.