UNCERTAINTY-SENSITIVE LEARNING AND PLANNING WITH ENSEMBLES

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a reinforcement learning framework for discrete environments in which an agent optimizes its behavior on two timescales. For the short one, it uses tree search methods to perform tactical decisions. The long strategic level is handled with an ensemble of value functions learned using TD-like backups. Combining these two techniques brings synergies. The planning module performs *what-if* analysis allowing to avoid short-term pitfalls and boost backups of the value function. Notably, our method performs well in environments with sparse rewards where standard $TD(1)$ backups fail. On the other hand, the value functions compensate for inherent short-sightedness of planning. Importantly, we use ensembles to measure the epistemic uncertainty of value functions. This serves two purposes: a) it stabilizes planning, b) it guides exploration.

We evaluate our methods on discrete environments with sparse rewards: the Deep sea chain environment, toy Montezuma's Revenge, and Sokoban. In all the cases, we obtain speed-up of learning and boost to the final performance.

1 INTRODUCTION

The model-free and model-based approaches to reinforcement learning have a complementary set of strengths and weaknesses. While the former offers good asymptotic performance, it suffers from poor sample complexity. In contrast, the latter usually needs significantly less training samples, but often fails to achieve state-of-the-art results on complex tasks (primarily attributed to models' imperfection). In environments with sparse rewards the value estimation has large variance, which may hinder the learning process, especially in its early stage.

In this work we propose a new approach blending model-based, model-free methods and utilizing risk-sensitivity information. It enhances explorations and mitigates the problems of sparsity in two ways, a) the planning module effectively shortens time horizon allowing to capture a denser learning signal, b) both value estimation and its uncertainty is used for bootstrapping and guiding exploration. We deal with environments where the optimal trajectory does not visit the same state twice, and we assume access to the perfect model. Our planning module is a Monte-Carlo Tree Search (MCTS)-like algorithm with several improvements utilizing this access to the model. Although they are not crucial to our method, we think they might be of general interests. Using MCTS is not necessary, in principle virtually any planning method could be used (e.g., Levin tree search, Orseau et al. (2018b)).

Handling and using the uncertainty of value functions is an important topic. We explore risk-sensitivity measures based on moments and relative majority vote. Secondly, we test ideas inspired by the Thompson sampling and Osband et al. (2016). Finally, in some experiments, we use Monte-Carlo tree search methods, with their notions of exploration. We found the interaction of these methods quite interesting and important.

The rest of the paper is organized as follows. In the next subsection we provide an overview of related topics. In Section 2 we present and discuss our method. This is followed by a section with experiments and passing to conclusions. We provide code to our work `https://github.com/learningandplanningICLR/learningandplanning` and a dedicated website `https://sites.google.com/view/learn-and-plan-with-ensembles` with more details and movies.

1.1 RELATED WORK

The ideas of mixing model-based and model-free learning were perhaps first stated explicitly in Sutton (1990). Many approaches followed. For example in the groundbreaking series of papers Silver et al. (2017), Silver et al. (2018) culminating in AlphaZero the authors have developed an elaborated system that plans and performs model-free training to master the game of Go (and others). Similar ideas were studied also in Anthony et al. (2017).

Constructing neural network models which would incorporate uncertainty in a principled Bayesian way has proven to be challenging and still unresolved. A promising new results using ensembles include Osband et al. (2018; 2017), Lakshminarayanan et al.. Ensembles of models were also successfully used improve model-based RL training, see Kurutach et al. (2018), Chua et al. (2018), and the references therein.

Perhaps, the work which is closest to ours is Lowrey et al. (2018). They argue, that an agent with limited computational resources in complex environment, needs both to plan and learn from the incoming stream of experience. They propose a system POLO, which plans using MPC Camacho & Bordons (2007) method and learns value function. Importantly, their value function is modeled by an ensemble of value functions which are aggregated using 'log-sum-exp formula' (Lowrey et al., 2018, 6), which roughly speaking corresponds a weighted sum of their mean and variance. They show experimentally that this approach lead to improvements in various tasks, notably in training of humanoid.

Another work similar to ours is Guo et al. (2014), in which the authors use MCTS in the role of 'an expert' from which a neural policy is learn using the dagger algorithm. The basic difference is that Guo et al. (2014) uses a classical MCTS without value function nor ensembles.

There are a number of works, which blend planning and learning into a end-to-end architectures, including Value Prediction Networks (Oh et al. (2017)) or TreeQN (Farquhar et al. (2017)). A recent work on model based Atari Kaiser et al. (2019) has showed possibility of sample efficient reinforcement learning with an explicit visual model. Gu et al. (2016) uses model-based at the beginning phase of training and model-free methods in 'fine-tuning', that is exactly where they are expected to excel. There is a line of work which attempts to learn planning module, some of them including 'imagination' modules Pascanu et al. (2017), Racanière et al. (2017) or mimicking a general scheme of MCTS-like algorithm into neural architecture Guez et al. (2019).

Our paper is also related to body of work related to exploration. Fundamental results in this area concern the multi-arm bandits problem, see Lattimore & Szepesvári (2018). Methods developed in this area have been successfully applied in planning algorithms, see Kocsis et al. (2006) and Silver et al. (2017; 2018). Another set of methods have been developed in an attempt to solve notoriously hard Montezuma's Revenge, see for example Go-Explore Ecoffet et al. (2019) or self-imitation learning techniques Guo et al. (2019). Another particularly interesting way of dealing with exploration is Hindsight method, see Andrychowicz et al. (2017).

2 METHOD DESCRIPTION

From a high-level perspective, our method alternates between data collection, which uses a planner, and learning, which optimizes a given architecture, see Algorithm 1. In this section we describe the aforementioned key components: planner, learning and architecture.

The method does not rely on a particular form of a planner. Having said that, in this work we focused on a version of MCTS similar to AlphaZero (Silver et al. (2018)). The method is novel in the sense that it introduces a uncertainty-sensitive policy (for tree traversal and action choice), as well as guided search which exploits the form of an optimal strategy (an idea which lies at the very heart of dynamic programming).

Algorithm 1 Learning and planning with ensembles

Input: initial parameters of value function ensemble θ_i, $i = 1, \ldots, K$, empty replay buffer $\mathcal{D}$
repeat
  Reset environment and initialize starting state s.
  while s is not terminal **do** ▷ Planning stage
    Initialize root ▷ Possibly soft-penalize loop
    while within computational budget **do** ▷ dubbed MCTS steps, e.g. 10
      Start from the root and move down the tree ▷ Tree-policy
      Expand leaf ▷ Using ensemble value network
      Backpropagate ▷ Possibly penalize a dead-end
    end while
    Choose next action and next state s ▷ Final policy, possibly penalize 1-cycles
  end while
  Evaluate episode ▷ Use e.g. factual or bootstrapped rewards
  Add the episode to replay buffer ▷ Possibly assigning masks
  for however many updates **do** ▷ Learning stage
    Randomly sample batch B of states and values ▷ Possibly also masks
    Use B to update ensemble value function with gradient descent (e.g. RMSProp)
  end for
until convergence

The approach is inspired by Osband et al. (2018) and Lowrey et al. (2018). The former introduces a way of modeling uncertainty via ensembles it was shown how ensembles can be viewed as samples from approximate posterior distribution (given the data gathered so far). The latter introduced a uncertainty-sensitive view into the planning. Consequently, we will consider the following strategies:

$$a^*(s) := \arg\max_a \mathbb{E}_{\theta\sim\Theta}\left[\phi_a(\widehat{\mathbf{Q}}_\theta)\right], \quad \widehat{\mathbf{Q}}_\theta := \left(\widehat{Q}_\theta(s,a')\colon a' \in \mathcal{A}\right). \tag{1}$$

where $\mathcal{A}$ is the action space, $\phi_a : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}$ is a uncertainty measure, and $\widehat{Q}_\theta$ is an estimator of Q-function. We take expectation over an ensemble of estimators using posterior distribution Θ. Lowrey et al. (2018) considered $\phi_a(x) = e^{\kappa x_a}$ for $x \in \mathbb{R}^{|\mathcal{A}|}$ and $\kappa > 0$. In this paper we consider the following ϕ_a:

- $\phi_a(x) = x_a + \kappa x_a^2$, $\kappa > 0$. This includes second moments. This can be easily generalized to include variance, standard deviation and exploration bonuses.
- we propose also
$$\phi_a(x) = 1\left(\arg\max_{a'} x_{a'} = a\right) \tag{2}$$
Contrary to the other cases this ϕ_a depends not only on marginal values of its input, but the whole input (i.e. the estimator vector $\widehat{\mathbf{Q}}_\theta$). It leads to a rule resembling optimal Bayes classifier form, i.e. the one which chooses a minimizing $\mathbb{P}(a^*_\theta(s) \neq a)$. Such a rule is also known as the relative majority measure.

Apart for the above mechanisms in some experiment we also use the Thomson sampling in a similar fashion to Osband et al. (2016). In the formulation of equation 1 this can be understood as "sub-sampling" from Θ.

In our MCTS planner we introduced model-based mechanism which enhances learning. Namely, while planning for the next action we unroll several rollouts, during this process we avoid entering perviously visited vertices. This might lead to a situation when no-vertex is available, which we dubbed as dead-end and penalize. Another similar mechanism avoids entering the previously visited states on the whole episode, which we call 1-loops. We found this mechanism to be beneficial and is able to learn even in sparse rewards scenarios (see "Learning solution for single boards" paragraph in Section 3.3).

During the learning phase, a batch is drawn from a replay buffer and the ensemble are optimized to minimize l_2 distance from sampled targets. In this paper we consider two ways of calculating targets:

a) values accumulated in tree nodes during planning phase (called *bootstrap*), b) (discounted) values of rewards collected in an episode (called *factual*). Our replay buffer also records for episodes, whether they are solved (i.e. those, which ended with positive final reward) or not. In our typical experiment batches are composed to contained a fixed ratio of transition from solved and unsolved episodes. In some experiments in training we use mask similar to the on in Osband et al. (2018).

In this work we typically implement ensembles as a set of neural networks (we also experiment with mult-headed architectures sharing some lower layers). We use both architectures with and without random priors Osband et al. (2018), for details see Appendix A.

3 EXPERIMENTS

In this section we provided experimental evidence to show that using risks measures are useful. We chose three environments, Deep-sea, toy Montezuma's Revenge and Sokoban.

In each case we use an MCTS planner with the number of passes equal to 10 (see line 6 of Algorithm 1). We consider this number to be a rather small for MCTS-like planning methods, still we observed that is is big enough to improve ensemble value functions.

We utilize various neural network architectures, see Appendix A. In most cases we measure uncertainty using either variance or standard deviation with an exception of Sokoban wih randomly generated board, where voting was used, see equation 2.

3.1 DEEP-SEA

Deep-sea environment was introduced in (Osband et al., 2018, Section 4) and later included in Osband et al. (2019) as a benchmark for exploration under the name. The agent is located on a $N \times N$ grid, $N \in \mathbb{N}$ starting at $(0, 0)$. In each timestep its y-coordinate is increased, while x is

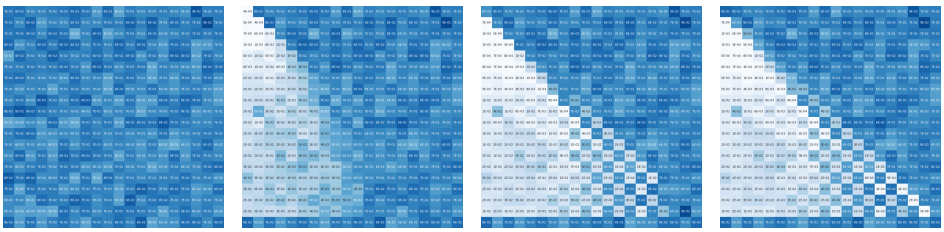

Figure 1: The heatmaps of standard deviations of values predicted by ensembles for states of the Deep-sea environment. High values are marked in blue and low in white. At the beginning of training, left picture, the standard deviation is high for all states. Gradually it is decreased in the states that have been explored. Finally, in the right, the reward state is found. Note that the upper-right part of the board is unreachable.

controlled. The agent issues actions $\{-1, 1\}$, which according to a prescribed mask are translated to one step "left" or one step "right" (decreasing, if possible, or increasing x respectively). For each step "right" the agent is punished with $0.01/N$. After N steps the game ends and the agent receives reward $+1$ if and only if it reaches position (N, N). The aforementioned mask is randomized at each field at the beginning of training (and kept fixed). Such a game is purposely constructed so that naive random exploration schemes fail already for small N's. Indeed, a random agent has chance $(1/2)^N$ of reaching the goal even is it disregards misleading rewards for "right" steps. On Figure 1 one can observe how the exploration progresses. On Figure 2 one can see comparison of non-ensemble models, ensemble model with Thomson sampling but without uncertainty benefit $\kappa = 0$ and our final ensemble model with exploration the bonus for uncertainty $\kappa > 0$.

3.2 TOY MONTEZUMAS REVENGE

Toy Montezuma's Revenge is a navigation maze-like environment. It was introduced in Roderick et al. (2018) to evaluate ideas of using higher-level abstractions in long-horizon problems.

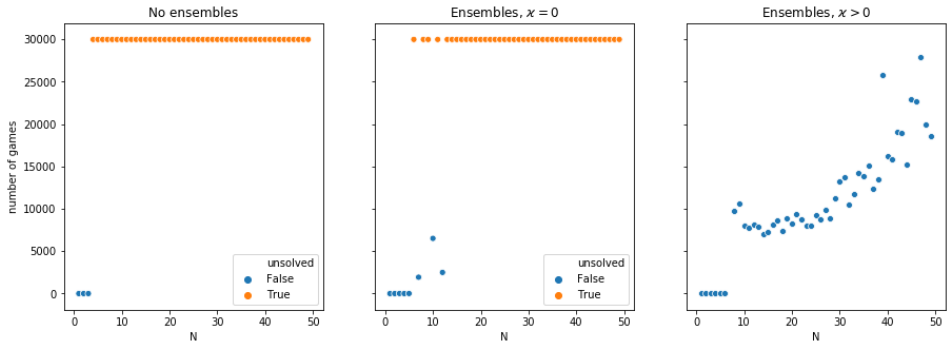

Figure 2: Comparison of number of episodes needed to solve the deep-sea environment with given grid size N. Orange dots marks trials which were unable to solve problem in 30000 episodes.

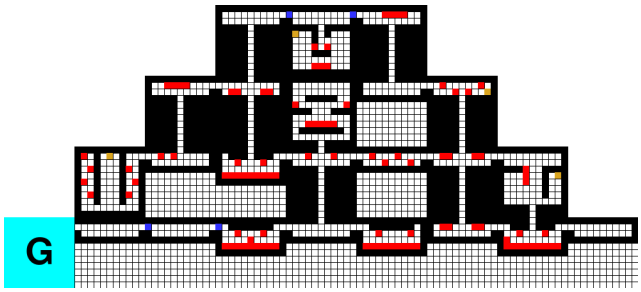

Figure 3: The biggest Montezuma's Revenge map, consisting of 24 room. The goal is to reach the room marked with G. The agent needs to avoid traps (marked in red) and pass through doors (marked in blue). Doors are open using keys (marked in yellow). (best view in color)

While its visual layer is greatly reduced it retains much of the exploration problems of the original Motemzuma's Revenge. This makes it a useful test environment for exploration algorithms, for example it was used in a recent self-imitation approach of Guo et al. (2019). From the presets of available rooms[1], in most of our experiments we work with the biggest map with 24 rooms, see Figure 3. In order to concentrate on the evaluation of exploration we chose to work with sparse rewards. The agent gets reward 1 only if it reaches the treasure room , otherwise the episode is terminated after 300 steps. Clearly, any simple exploration technique would fail in this case (we provide some baselines in Table 1). Guo et al. (2019) benchmarks PPO, PPO with self imitation learning (PPO+SIL), PPO with count based exploration bonus (PPP+EXP) and their new technique (DTSIL). Only the last new technique consistently is able to solve 24 room challenge, with PPO+EXP occasionally reaching this goal. Our method based on ensembles solves this exploration challenge even in a harder sparse reward case.[2] Our results are summarized in Table 1. We have three setups: 'no-ensemble', 'ensemble, no std', 'ensemble, std'. In the first case, we train using Algorithm 1 with single network, neural-network. In the second case, for each episode we sample 10 ensembles to be used and the MCTS is guided but their mean. In the final third case, we follow the same protocol but we add to the mean the standard deviation of the ensembles. In our experiments we observe that 'no-ensemble' often in 30 out of 43 cases does not leave behind the first room. Ensembles without explicit exploration bonus perform slightly better. Finally we observe that the setup using ensembles behaves very well.

Further experimental details and the network architecture are presented in Appendix A and B.

Setup	win-ratio (no. seeds)	av. visited rooms
no-ensemble	0 (43)	4.8
ensemble, no std	2 (40)	5.8
ensemble, std	35 (37)	17.1

Table 1: Result for toy Montezuma's revenge. We report the number of successful runs and the number of seeds of network initialization. We also show the average number of visited rooms, which is a proxy for the learning progress.

3.3 SOKOBAN

Sokoban is an environment known for its combinatorial complexity. The agent's

[1]We use a slightly modified code from `https://github.com/chrisgrimm/deep_abstract_q_network`

[2]DTSIL build on the intermediate partial solutions, which are ranked according to their reward, thus we suspect it would fail in the sparse reward case.

goal is to push all boxes to the designed spots, see Figure 4. Apart for the navigational factor, the difficulty of this game is greatly increased by the fact that some actions are irreversible. An archetypal example is pushing a box into a corner, though there are multiple less obvious cases. This is further confirmed by the fact that it is NP-hard, see e.g. Dor & Zwick (1999). Due to these challenges the game has been considered as a testbed for reinforcement learning and planning methods. The boards can be randomly procedurally generated and parameterized by their size and number of boxes.[3] Exploration problem in Sokoban can be understood at two levels: a) it appears on one board, b) at meta-level exploration is needed to find a solution common to all boards.

In many RL approaches to Sokoban, it is used with intermediate rewards e.g. for pushing a box into a designed spot. In this work we use *sparse setting*. The agent is rewarded only when all boxes are in place. We conducted three lines of experiments using ensembles: *a) learning solution to randomly generated boards, b) learning solution to single boards, c) transfer and learnable ensembles*.

In our experiment we use Sokoban with board of size $(10, 10)$ and four boxes. We found that the maximal length of an episode influences the results significantly. We use 200 steps in the experiments with learning generalized value function and 100 in others.

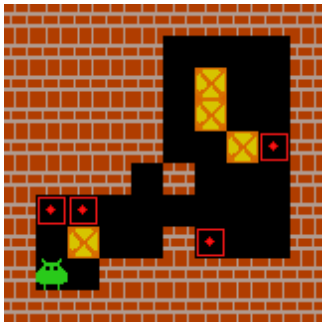

Figure 4: Example $(10, 10)$ Sokoban board with 4 boxes. Boxes (yellow) are to be pushed by agent (green) to designed spots (red). The optimal solution has 37 steps.

Learning solution to randomly generated boards In this experiment we measure the ability of our approach to solve randomly generated Sokoban boards. We measure the performance of the agent by computing the win rate on last 1000 games. In this experiment we use an ensemble value function using relative majority voting as formalized in equation 2.

Relative majority voting takes into account the disagreement of ensembles when it comes to the final outcome, not only the disagreement in assessment of particular action. After 80000 games it reaches 85% win rate compared to 76% of an agent not using ensembles, see Figure 5. As a proxy measure of ensemble disagreement, we also present the mean standard deviation of value function across episodes (red curve in Figure 5). It suggests that the better the ensembles get, the more the agree on the values assigned to Sokoban states. It also indicates that ensembles reduce the amount of exploration the better they get (measured via win rate).

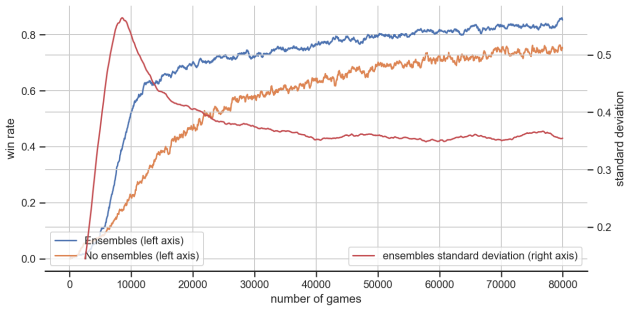

Figure 5: Learning curve (left axis) and standard deviation of ensemble values (right axis) in Sokoban. The plots are functions of the number of games. (best view in color)

Learning solution to single boards In this experiment we measure the extent in which our methods can plan and learn on single board of Sokoban. We note that this setting differs substantially from the one in the previous paragraph. While learning a generalized value function is typically a harder task, in our case it has also positive impact on the training process. There are significantly many boards which can be easily solved even at the starts (we tested that in our setting MCTS with randomly initialized network solves $\approx 0.7\%$ of boards), which can be used as positive examples. This leads to emergent implicit curriculum. In our experiments with single boards, we used setups similar to the one in Montezuma's Revenge. We observes that the setup with ensembles solves $\approx 65\%$ compare to

[3]There is a subtlety, as the difficulty of board is heavily influenced by the generation protocol. We follow the one from Racanière et al. (2017).

$\approx 50\%$ the standard one network scenario. To ensure statistical significance in both of the cases we performed ≥ 250 experiments thus obtaining standard deviations $\leq 2.5\%$. The result of the standard setup might seem surprisingly good, taking into account the sparse reward. This follow by the loop avoidance described in Section 2. In this experiment if during planning the agent finds itself in a situation from which it cannot find a novel state it is punished (with -2). This value is propagated to the learning process creating another form of exploration.

Transfer and learnable ensembles Generating any new board can be seen as a cost dimension along with sample complexity. This quite naturally happens in meta-learning problems. We tested how value functions learned on small number of boards perform on new previously unseen ones. We used the following protocol, we trained value function on fixed number of 10 games. To ease the training we used relabeling akin to **??**[4]. We evaluate these models on other boards. It turns out that they are typically quite weak, arguably it is not very surprising as solving 10 boards does not give much chance to infer 'the general' solutions. In the second phase we use ensembles of the models. More precisely, we calculate the values of $n = 2, 3$ models and aggregate it using a small neural networks with one fully connected hidden layer. This network is learnable and trained using the standard setup. We observe that the quality of such ensemble increases with the number of components as summarized in Table 2. We observed high variability of the results over seeds, this is to be expected as board in Sokoban significantly vary in difficulty. We also observe that maximal results for transfer increase with the number of value function, being approximately 10%, 11% and 12%. This further supports the claim that ensembling may lead to improved performance. In 5-layer experiment we use a network with 5 hidden cnn layers, see details in Appendix A, we compare this with an analogous 4 layers network. In the latter case, we obtain weaker result. We speculate this might be due to the fact that smaller architecture is easier to overfit.

Architecture	Random	Trans. 1	Trans. 2	Trans. 3
5-layers	0.7%	4.9%	7.1%	$8.5\ \%$
4-layers	0.7%	4.3%	5.6%	7.3%

Table 2: Results of transfer experiments. We test transfers from one value function (Trans 1) and transfer from ensembles of 2 and 3 value functions (Trans 2 and Trans 3). In the later two cases the aggregation of values is learned. The results are averaged over 20 seeds.

CONCLUSIONS AND FURTHER WORK

In this paper, we introduced a reinforcement learning method that blends planning, learning, and using information about uncertainty to boost exploration. We verified experimentally that such a setup is useful enabling deep exploration for time horizons spanning for even hundreds of steps.

We believe that this opens promising future research directions. There are many choices for ensembles designs some of which we tested, but there are still many others. At the moment, selecting a proper aggregation method seems somewhat problem-specific. The ultimate goal should be finding more general methods. Such developments would be a step towards deep Bayesian learning.

In our work, we used MCTS with a perfect model of deterministic environments. It would be interesting to consider problems requiring the use of learned, imperfect models. This is more demanding, though uncertainty methods might be of use, this time to avoid model imperfections. Equally important would be tackling stochastic environments. This might be considerably more difficult as such a task requires disentangling epistemic (studied in this work) and aleatoric (coming from the environment) uncertainties.

We focused our attention on the Monte-Carlo tree search, there are many other known planning algorithms, which could benefit from utilizing the information about uncertainty. In some initial experiments we obtained promising but yet inconclusive result using the Levine tree search (see Orseau et al. (2018a)). Another tempting direction is training both value and policy, akin to methods of Silver et al. (2017).

[4]More precisely, for a failing trajectory we choose a random time-step and shift the target spots so that they match the current location of boxes. We note that although this operation requires the knowledge of the game mechanics (i.e. its perfect model) it is used only in this phase

The case of Sokoban cast a problem into meta-learning and continual learning grounds. Using measures of uncertainty might enable a learning system to adapt to a changing environment. In an archetypical case, this might be obtained by choosing from ensemble a model (a skill) which is useful at the moment and understanding situations that such a model is not yet present.

REFERENCES

Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 5048–5058, 2017. URL http://papers.nips.cc/paper/7090-hindsight-experience-replay.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *NIPS*, 2017.

E. F. Camacho and C. Bordons. *Model Predictive control*. Advanced Textbooks in Control and Signal Processing. Springer London, London, 2007. ISBN 978-1-85233-694-3. doi: 10.1007/978-0-85729-398-5.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS 2018*, pp. 4759–4770, 2018.

Dorit Dor and Uri Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019. URL http://arxiv.org/abs/1901.10995.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree planning for deep reinforcement learning. *CoRR*, abs/1710.11417, 2017.

Shixiang Gu, Timothy P. Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy P. Lillicrap. An investigation of model-free planning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 2464–2473, 2019. URL http://proceedings.mlr.press/v97/guez19a.html.

Xiaoxiao Guo, Satinder P. Singh, Honglak Lee, Richard L. Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *NIPS*, 2014.

Yijie Guo, Jongwook Choi, Marcin Moczulski, Samy Bengio, Mohammad Norouzi, and Honglak Lee. Efficient exploration with self-imitation learning via trajectory-conditioned policy. *CoRR*, abs/1907.10247, 2019. URL http://arxiv.org/abs/1907.10247.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019. URL http://arxiv.org/abs/1903.00374.

Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1, 2006.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *CoRR*, abs/1802.10592, 2018.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS 2017*.

Tor Lattimore and Csaba Szepesvári. Bandit algorithms. *preprint*, 2018.

Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *CoRR*, abs/1811.01848, 2018.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *NIPS*, 2017.

Laurent Orseau, Levi Lelis, Tor Lattimore, and Theophane Weber. Single-agent policy tree search with guarantees. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 3205–3215, 2018a. URL http://papers.nips.cc/paper/7582-single-agent-policy-tree-search-with-guarantees.

Laurent Orseau, Levi Lelis, Tor Lattimore, and Théophane Weber. Single-agent policy tree search with guarantees. In *Advances in Neural Information Processing Systems*, pp. 3205–3215, 2018b.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *NIPS*, 2016.

Ian Osband, Daniel Russo, Zheng Wen, and Benjamin Van Roy. Deep exploration via randomized value functions. *CoRR*, abs/1703.07608, 2017.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *NeurIPS*, 2018.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepezvári, Satinder Singh, Benjamin Van Roy, Richard S. Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. *CoRR*, abs/1908.03568, 2019. URL http://arxiv.org/abs/1908.03568.

Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sébastien Racanière, David P. Reichert, Theophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *CoRR*, abs/1707.06170, 2017.

Sébastien Racanière, Theophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *NIPS*, 2017.

Melrose Roderick, Christopher Grimm, and Stefanie Tellex. Deep abstract q-networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pp. 131–138, 2018. URL http://dl.acm.org/citation.cfm?id=3237409.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 1144:1140–1144, 2018.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ML*, pp. 216–224. Morgan Kaufmann, 1990.

A ARCHITECTURES

Deep-sea For Deep-see environment we encode state as one hot vector of size N^2, and learn simple linear transformation for value estimation.

Toy Montezuma Revenge Observation is represented as tuple containing current room location, agent position within the room and status of all keys and doors on the board. To estimate value we use fully-connected neural networks with two hidden layers consisting 50 neurons each.

Single-board Sokoban Observation has shape (10, 10, 7) where first two coordinates are spatial and last one encodes type of field (e.g. box, target, agent, wall). To estimate value we flatten the observation and apply fully-connected neural networks with two hidden layers consisting 50 neurons each.

Multiple-boards Sokoban Here we use same observation as in single-board problem. To generalize between different Sokoban levels we use convolutional neural network with 5 hidden layers consisting 64 channels each.

B TRAINING DETAILS

We considered two objectives when training value function from recorded episodes.

Bootstap method was used in our Deep-see, Toy-Montezuma-Revenge and single-board Sokoban experiments.

Parameter	Value
Number of MCTS passes	10
Ensemble size	20
Learning rate	0.00025
Optimizer	RMSProp
Discounting γ	0.99
κ	3

Table 3: **Hiperparameters** Default values used for our experiments. Exceptions is $\kappa = 50$ for Deep-see environment.

.