

Robust and Efficient Planning using Adaptive Entropy Tree Search

Piotr Kozakowski¹ Mikołaj Pacek¹ Piotr Miłoś²

Abstract

In this paper, we present the Adaptive Entropy Tree Search (ANTS) algorithm. ANTS builds on recent successes of maximum entropy planning (Xiao et al., 2019; Dam et al., 2020) while mitigating its arguably major drawback - sensitivity to the temperature setting. We endow ANTS with a mechanism, which adapts the temperature to match a given range of action selection entropy in the nodes of the planning tree. With this mechanism, the ANTS planner enjoys remarkable hyperparameter robustness, achieves high scores on the Atari benchmark, and is a capable component of a planning-learning loop akin to AlphaZero. We believe that all these features make ANTS a compelling choice for a general planner for complex tasks.

1. Introduction

Planning is one of the main approaches to sequential decision making. A particularly successful method of planning is Monte Carlo Tree Search (MCTS) - an algorithm based on iteratively expanding a tree of possible decisions using randomized simulations. MCTS is very versatile, finding applications ranging from playing complex strategic games (Lee et al., 2009), through testing security systems (Tanabe et al., 2009), physics simulations (Mansley et al., 2011) and production management (Chaslot et al., 2006), to creative content generation (Mahlmann et al., 2011). Its modular structure allows a wide range of possible modifications, and enables easy adaptation to the task at hand.

A very promising direction of research is the combination of MCTS with deep learning. It has already powered one of the major breakthroughs in the history of Artificial Intelligence: AlphaZero (Silver et al., 2018), a system combining planning and learning to surpass the world champions in

one of the most complex strategic games - go. Planning and learning can be thought of as two complementary approaches to utilizing computation to generate intelligent behavior. On the one hand, planning can discover high-quality decision sequences through search, but falls short in large state spaces. On the other hand, learning can leverage patterns in the data to discover useful state representations, but cannot make use of additional online computation time. Utilizing learning to discover useful state representations can greatly reduce the search space and guide planning to make it significantly more efficient.

The Principle of Maximum Entropy has been shown to correctly model collective human behavior (Ziebart, 2010; Hernando et al., 2013), as well as robust decision making in face of reward uncertainty and adversity (Eysenbach & Levine, 2019). Encoding uncertainty as to the true objective into an intelligent agent is thought to be a promising approach to designing AI systems that align with human goals (Hadfield-Menell et al., 2017), which makes research on maximum entropy methods especially relevant in contemporary times. The Principle of Maximum Entropy has also inspired some very successful reinforcement algorithms (Haarnoja et al., 2018).

Extending the maximum entropy approaches to planning seems like a natural direction, and indeed - Xiao et al. (2019) and Dam et al. (2020) place maximum entropy methods in the framework of Monte Carlo Tree Search to derive successful planning algorithms for complex visual tasks. We build upon these works and present a robust and efficient planning agent, capable of solving a range of diverse tasks with the same settings. We empirically evaluate the design choices of our method, and finally demonstrate its ability to be complemented with learning, to establish a standalone, end-to-end planning and learning system.

2. Background

2.1. Reinforcement Learning and Planning

Reinforcement learning (RL) focuses on solving sequential decision problems formalized as Markov decision processes (MDPs). We consider a deterministic, discounted, finite-horizon MDP, which is defined as a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where \mathcal{S} is the set of states, \mathcal{A} is the set of actions, \mathcal{P} :

¹Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland ²Institute of Mathematics, Polish Academy of Sciences, Poland. Correspondence to: Piotr Kozakowski <p.kozakowski@mimuw.edu.pl>.

$\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function and $\gamma \in [0, 1)$ is the discount factor. We define a *policy* $\pi(a|s)$ as a probability distribution over the action to take, conditioned on the current state. For a given policy π , we define the *state value function* $V^\pi(s) = \mathbb{E}_{(s_t, a_t)_{t=0}^\infty} [\sum_{t=0}^\infty \gamma^t \mathcal{R}(s_t, a_t)]$, where $s_0 = s$, $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} = \mathcal{P}(s_t, a_t)$, as the expected cumulative return obtained by π from state s . We define the *state-action value function* $Q(s, a) = \mathcal{R}(s, a) + \gamma V(s')$, where $s' = \mathcal{P}(s, a)$, as the expected cumulative return obtained by executing action a in state s and then continuing using π .

RL methods aim to find the optimal policy π^* by *learning* from trajectories sampled from the MDP. The optimal state value function $V^* = V^{\pi^*}$ and state-action value function, also called Q -function, $Q^* = Q^{\pi^*}$ satisfy a recursive relationship known as the Bellman optimality equations (Sutton & Barto, 2018, Section 3.6): $V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a)$, $Q^*(s, a) = \mathcal{R}(s, a) + \gamma V^*(s')$, where $s' = \mathcal{P}(s, a)$. The optimal policy is the *greedy policy* with respect to the optimal state-action value function: $\pi^*(a|s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$. Model-based RL methods additionally assume access to a *model* of the MDP $\mathcal{P}' : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, which they typically use to consider multiple different rollouts from a given state, in a process known as *planning*. In this work, we assume access to the perfect model of the MDP, so $\mathcal{P}' = \mathcal{P}$.

2.2. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a class of planning algorithms for discrete action spaces. MCTS iteratively builds a planning tree, where nodes correspond to states and edges correspond to (state, action) pairs. MCTS estimates values of the nodes by aggregating multiple paths from the root to a leaf. MCTS consists of four phases, repeated in a loop (Browne et al., 2012):

1. **Selection**, in which a path from the root to a leaf is selected.
2. **Expansion**, in which a successor state of the selected leaf is added to the tree as a new leaf.
3. **Simulation**, in which the value of the new is estimated using random rollouts.
4. **Backpropagation**, in which the value estimates of nodes on the path are updated, starting from the bottom.

Numerous improvements to MCTS have been developed over the years. Notably, Silver et al. (2018) supplement MCTS with neural networks. In the selection phase, the next node is selected to maximize

$$\text{PUCT}(s, a) = Q(s, a) + c \hat{\pi}(a|s) \frac{\sqrt{N(s)}}{N(s, a)} \quad (1)$$

from the current node s , where c is an exploration parameter, $\hat{\pi}(a|s)$ is a *prior network*, $N(s)$ is the number of visits in state s and $N(s, a)$ is the number of times action a has been executed in state s . In the simulation phase, the value of a leaf s is estimated using a *value network*: $\hat{V}(s)$. The prior network is trained to predict the distribution of actions selected during planning: $\frac{N(s, a)}{N(s)}$. The value network is trained to estimate the empirical state values $V(s)$.

In this work, we consider an expansion phase which adds all successors of a given node to the tree at the same time. Those new leaves are evaluated using a Q -network $\hat{Q}(s, a)$. This is an optimization suitable for problems, where computing state transitions is expensive. The Q -network is trained to estimate the empirical state-action values $Q(s, a)$.

2.3. Maximum Entropy RL

Maximum entropy RL is a popular approach to exploration in RL. It has recently given rise to a number of successful algorithms (Haarnoja et al., 2017; 2018; Hausman et al., 2018). It augments the RL objective with an entropy regularization term, rewarding uncertainty of the policy and thus promoting exploratory behaviors:

$$J(\pi) = \mathbb{E}_{(s_t, a_t)_{t=0}^\infty} \left[\sum_{t=0}^\infty \gamma^t [\mathcal{R}(s_t, a_t) + \tau \mathcal{H}(\pi(\cdot|s_t))] \right] \quad (2)$$

where \mathcal{H} is Shannon entropy and τ is a temperature parameter, regulating the amount of exploration. We will denote the solution to the entropy-regularized RL objective as $\pi^* = \operatorname{argmax}_\pi J(\pi)$.

Haarnoja et al. (2017) solve this objective using *soft Q-iteration* - fixed-point iteration on:

$$\begin{aligned} Q_{\text{sqi}}(s, a) &\leftarrow \mathcal{R}(s, a) + \gamma V_{\text{sqi}}(s') \\ V_{\text{sqi}}(s) &\leftarrow \tau \log \int_{\mathcal{A}} \exp \left(\frac{1}{\tau} Q_{\text{sqi}}(s, a) \right) da, \end{aligned} \quad (3)$$

which results in $Q_{\text{sqi}} = Q_{\text{sqi}}^*$ and $V_{\text{sqi}} = V_{\text{sqi}}^*$. The optimal policy can then be recovered as:

$$\pi^*(a|s) = \exp \left(\frac{1}{\tau} (Q_{\text{sqi}}^*(s, a) - V_{\text{sqi}}^*(s)) \right) \quad (4)$$

Haarnoja et al. (2018) find π^* using *soft policy iteration* - fixed-point iteration on:

$$\begin{aligned}
 Q_{\text{spi}}(s, a) &\leftarrow \mathcal{R}(s, a) + \gamma V_{\text{spi}}(s') \\
 V_{\text{spi}}(s) &\leftarrow \mathbb{E}_{a \sim \pi(\cdot|s)}[Q(s, a) - \tau \log \pi_{\text{spi}}(a|s)] \\
 \pi_{\text{spi}}(a|s) &\leftarrow \frac{\exp(\frac{1}{\tau} Q_{\text{spi}}(s, a))}{\sum_{a'} \exp(\frac{1}{\tau} Q_{\text{spi}}(s, a'))},
 \end{aligned} \tag{5}$$

which results in $\pi_{\text{spi}} = \pi^*$.

Xiao et al. (2019) introduce MENTS: an adaptation of soft Q-iteration applied to MCTS. In the selection phase, consecutive nodes are chosen using the E2W sampling strategy. The sampling distribution is a weighted average between uniform and the softmax policy over Q-values:

$$\begin{aligned}
 \pi_{\text{e2w}}(a|s) &= (1 - \lambda_s) \pi_{\text{softmax}}(a|s) + \lambda_s \frac{1}{|\mathcal{A}|} \\
 \text{where } \pi_{\text{softmax}}(a|s) &\propto \exp\left(\frac{1}{\tau} Q(s, a)\right) \\
 \lambda_s &= \frac{\epsilon |\mathcal{A}|}{\log(N(s) + 1)}
 \end{aligned} \tag{6}$$

In the simulation phase, leaves are evaluated using a Q-network. In the backpropagation phase, node state values and Q-values are updated according to (3).

Dam et al. (2020) introduce E3W - a generalization of the E2W sampling strategy to different types of entropy. They derive two new planning methods using E3W: TENTS, based on Tsallis entropy, and RENTS, based on relative entropy.

It turns out, that maximum entropy RL methods are highly sensitive to the temperature parameter τ . It depends on the scale and frequency of the rewards, so typically requires careful tuning per-task. Furthermore, the agent receives higher rewards as it becomes better, so the optimal temperature can also change throughout training. Haarnoja et al. (2019) introduce a method of automatic adjustment of the temperature in the context of continuous control. They enforce a lower bound on the average entropy, turning the maximum entropy objective (Equation 2) into a constrained optimization problem:

$$\begin{aligned}
 \pi^* &= \operatorname{argmax}_{\pi} \mathbb{E}_{(s_t, a_t)_{t=0}^{\infty}} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \right] \\
 \text{s.t. } &\mathbb{E}_{s, a} [-\log \pi(a|s)] \geq \mathcal{H}
 \end{aligned} \tag{7}$$

3. Adaptive Entropy Tree Search

Now we describe our method, Adaptive Entropy Tree Search (ANTS). We follow the general template of MCTS, with the

addition of maximum entropy exploration and temperature adaptation.

3.1. Temperature Adaptation

In MCTS, we have access to the set of all states explored so far to select the best action in the current state. We label this set $\mathcal{S}_{\text{tree}}$. We consider how the entropies of the nodes corresponding to those states would change with different values of the temperature, to determine its optimal value. As such, our method can use different values of the temperature not only for different tasks and for different episodes of the same task, but can also dynamically adapt the temperature to changing conditions within a single episode, which provides additional flexibility. This feature may be useful in tasks consisting of multiple logical parts, that require different exploration strategies.

We have found that in our setting, imposing a lower bound on mean entropy, in a way similar to Haarnoja et al. (2019), is not enough to ensure proper exploration. The minimum values of the entropy can still be very low, which causes the planner to prematurely reject promising paths. Instead, we:

1. Minimize **mean deviation** of the node entropy from the specified range.
2. Enforce an **upper bound** \mathcal{H}_{max} for the entropy, in addition to the lower bound \mathcal{H}_{min} . This way the temperature cannot increase boundlessly.
3. Add a **regularizer** $\beta \log \tau$ to penalize high temperatures. This breaks ties between different values of temperature, that arise when all nodes have entropy within the desired range.

This results in the following objective function:

$$\begin{aligned}
 L(\tau) &= \sum_{s \in \mathcal{S}_{\text{tree}}} \max(\mathcal{H}_{\text{min}} - \mathcal{H}_{\tau}^s, \mathcal{H}_{\tau}^s - \mathcal{H}_{\text{max}}, 0) + \beta \log \tau \\
 \text{where } \mathcal{H}_{\tau}^s &= \mathcal{H}(\pi_{\tau}(\cdot|s)) \\
 \pi_{\tau}(a|s) &\propto \exp\left(\frac{1}{\tau} Q(s, a)\right)
 \end{aligned} \tag{8}$$

We calculate it over the set of states in the current planning tree $\mathcal{S}_{\text{tree}}$ and minimize using the Brent algorithm (Brent, 1973, page 73).

3.2. Temperature Smoothing

At the beginning, we set the temperature to a constant, high value τ_0 . This is meant to ensure good estimation of the

initial Q-values in the tree, that will be used to determine the optimal temperature in the later stages. Every m planning passes, we optimize the temperature using the procedure described in subsection 3.1 to obtain τ_i , where i is the index of the planning pass. We smooth the resulting temperature using an exponential moving average $\hat{\tau}_i$ of decay α . We use the smoothed temperature to guide node selection and to backpropagate values in the later planning passes. To be able to quickly adapt the order of the temperature, we calculate the moving average in log-space:

$$\log \hat{\tau}_{i+1} = \alpha \log \hat{\tau}_i + (1 - \alpha) \log \tau_{i+1} \quad (9)$$

The moving average serves a dual purpose: in addition to making the temperature changes smoother, it allows the temperature to gradually decrease from τ_0 , as the planner progressively improves its estimates of node Q-values.

3.3. Soft Policy Iteration

In the backpropagation phase, we use the soft policy iteration update rule (5). This is in contrast to Xiao et al. (2019) and Dam et al. (2020), who use the soft Q-iteration rule (3) - as we have found, soft policy iteration performs either better, or comparatively well, across the evaluated tasks. One possible explanation is that soft policy iteration adds pseudo-rewards based on the entropy of the policy to the state values, which causes the planner to seek out nodes with high entropy in the tree. Such high-entropy nodes are the ones where the planner is the most uncertain about what is the optimal action, so they will benefit the most from more computational budget.

We additionally use a simple reward shaping mechanism to make the pseudorewards less than or equal to zero, by subtracting the entropy of the uniform distribution: $\log |\mathcal{A}|$. Such shaping does not change the optimal policy obtained at infinite computational budget, but prevents a depth-first exploration behavior. We have witnessed such behavior in case of sparse rewards, when the planner keeps deepening the same, single path, because it keeps receiving positive pseudorewards along it.

Our final backpropagation rule is:

$$\begin{aligned} Q_\tau(s, a) &\leftarrow \mathcal{R}(s, a) + \gamma V_\tau(s') \\ V_\tau(s) &\leftarrow \mathbb{E}_{a \sim \pi_\tau(\cdot|s)} [Q_\tau(s, a) + \mathcal{R}'_{\pi_\tau}(s, a)] \end{aligned} \quad (10)$$

$$\text{where } \mathcal{R}'_{\pi_\tau}(s, a) = -\tau \log \pi_\tau(a|s) - \tau \log |\mathcal{A}|$$

3.4. Greedy Selection

In the selection phase, we always choose the most under-explored action, i.e. the one with the highest mismatch

between the empirical visit distribution and the target distribution. This is in contrast to Xiao et al. (2019) and Dam et al. (2020), who sample actions from the target distribution. We have found that greedy selection consistently achieves higher performance across the evaluated tasks. We believe this to be because by always picking the most under-explored action, we can reach the target distribution faster. For instance, this way we never pick the over-explored actions, which would skew the empirical visit distribution even further from the target. We set our target distribution to be the softmax distribution over Q-values, π_τ . This results in the following selection rule:

$$\operatorname{argmax}_a \left[\pi_\tau(a|s) - \frac{N(s, a)}{N(s)} \right] \quad (11)$$

Conversely, Xiao et al. (2019) and Dam et al. (2020) set the target distribution to be π_{e2w} - a weighted average of the uniform distribution and the softmax distribution over Q-values (6), to ensure an accurate estimate of $Q(s, a)$ in the initial stages of exploration. In our experiments, gradually decreasing the temperature from a high value τ_0 as described in subsection 3.2 was enough to provide a good estimate.

4. Related Work

4.1. Reinforcement Learning

Recent years have brought significant advancements in the field of reinforcement learning, due to the use of powerful neural function approximators. Mnih et al. (2015) introduce DQN: Q-learning with the use of deep neural networks, leading to human-level performance on the tasks from Arcade Learning Environment (Bellemare et al., 2012). Hessel et al. (2018) combine multiple improvements to DQN, enabling superior performance across the ALE benchmark. Badia et al. (2020) introduce a method of parameterizing a family of policies at different points of the exploration-exploitation spectrum, leading to an algorithm which achieves superhuman performance on all 57 ALE tasks.

Model-based RL methods assume access to a model of the MDP - either specified from outside, or learned. The classical work of Sutton (1991) introduces a method of learning the model alongside the agent. The model is used to generate imaginary experience for RL training. Kaiser et al. (2019) apply a similar approach to the ALE benchmark, establishing state-of-the-art in performance in the low budget of 100K interactions with the environment. Ha & Schmidhuber (2018) present a method of planning in the latent space of a learned model. Hafner et al. (2019) plan by propagating gradients through the latent states of a learned model. Wang et al. (2019) provide a comprehensive empirical evaluation of model-based RL methods applied to continuous control. Those methods include MB-MPO (Clavera et al., 2018) -

meta-learning of the policy using experience generated by an ensemble of models, SVG (Heess et al., 2015) - learning policies by backpropagating value gradients through a stochastic model, and PETS (Chua et al., 2018) - planning by sampling trajectories from an ensemble of stochastic models.

4.2. Monte Carlo Tree Search

MCTS is a combination of the UCB strategy for multi-armed bandits with tree search planning (Kocsis & Szepesvári, 2006). Browne et al. (2012) provide a survey of methods in the MCTS family. Silver et al. (2018) introduce AlphaZero - MCTS powered with neural networks and self-play, to obtain a system achieving superhuman performance in complex strategic board games: chess, shogi and go. Schrittwieser et al. (2019) develop MuZero, an extension of AlphaZero with a learned model, and establish new state-of-the-art on ALE. Hamrick et al. (2020b) provide an extensive empirical analysis of MuZero, shedding new light on the role of planning in model-based RL.

Each phase of the MCTS algorithm (selection, expansion, simulation, backpropagation) can be replaced and improved, which gives rise to a plethora of methods in the MCTS family. Grill et al. (2020) find a connection between MCTS and policy optimization, which they use to derive a novel selection rule. Kozakowski et al. (2020) regulate the bias-variance trade-off by expanding multiple consecutive leaves along a path. Hamrick et al. (2020a) perform simulation (leaf evaluation) using neural networks trained via Q-learning. Miłoś et al. (2019) use ensembles of neural networks for uncertainty-aware simulation. Khandelwal et al. (2016) use the intuitions from TD- λ value estimation (Sutton, 1988) to derive new backpropagation rules for MCTS.

4.3. Maximum Entropy RL

The Principle of Maximum Entropy (PME) has shown success in modeling human behavior (Ziebart, 2010). Levine (2018) describe the maximum entropy RL problem as inference in a probabilistic graphical model, enabling its solution using the existing methods from the probabilistic inference toolkit. Ziebart et al. (2008) apply the PME to inverse reinforcement learning, and pose the problem of imitation learning as finding an optimal maximum entropy policy in the constructed MDP. Eysenbach & Levine (2019) analyze the maximum entropy RL problem and show its equivalence to classes of MDPs with uncertain or adversarial reward functions. (Nachum et al., 2017) establish a connection between policy-based and value-based RL under entropy regularization. Haarnoja et al. (2017) introduce Soft Q-Learning: a deep RL method derived from soft Q-iteration (3). Haarnoja et al. (2018) present Soft Actor-Critic (SAC): a method

based on soft policy iteration (5), achieving state-of-the-art performance in continuous control tasks. Hausman et al. (2018) establish a variational lower bound for the entropy pseudo-reward to learn an embedding space of transferable robotic skills. Haarnoja et al. (2019) derive a method of automatically adapting the temperature parameter of SAC. Xiao et al. (2019) use the maximum entropy framework to derive E2W: an optimal sampling strategy for the soft-max stochastic bandit problem, and introduce MENTS - an MCTS algorithm using this strategy in the selection phase. Dam et al. (2020) extend E2W to different types of entropy: Tsallis entropy, leading to the TENTS algorithm, and relative Shannon entropy, leading to the RENTS algorithm.

5. Experiments

We run our experiments on the same set of 22 Atari games as Dam et al. (2020). The experiments come in two flavors: for the first series of experiments, we follow a similar setup as Xiao et al. (2019) and Dam et al. (2020), and use Q-networks pretrained using DQN for leaf evaluation. In the second series, we train the Q-networks from in a closed loop, using Q-value targets calculated by the planner, in a similar fashion to Silver et al. (2018).

5.1. Pretrained Q-Networks

We first evaluate the performance of ANTS with pretrained Q-networks used for leaf evaluation. In this experiment, we want to measure the benefit from adaptive temperature tuning and other improvements implemented in ANTS. For this reason, we run each algorithm with the same set of hyperparameters across all tasks. This stands in contrast to Xiao et al. (2019); Dam et al. (2020), who tune τ and ϵ separately for each task. We set the temperature $\tau = 0.01$ for MENTS, $\tau = 1.0$ for TENTS and the exploration parameter $\epsilon = 0.1$ for both methods. τ has been selected from range $[0.01, 1.0]$, and ϵ from range $[0.1, 10.0]$, similarly to the authors of those algorithms. In ANTS, we set $\mathcal{H}_{\min} = 0.5$, $\mathcal{H}_{\max} = 1.0$, $\alpha = 0.9$ and $\beta = 0.001$. In all algorithms, we run 500 MCTS passes before choosing each action. We select an action with the maximum Q-value returned by the planner.

Similarly to Xiao et al. (2019); Dam et al. (2020) we use networks pretrained with DQN. For this purpose, we use the publicly available checkpoints from the Dopamine package (Castro et al., 2018). We use MENTS and TENTS as baselines. In those algorithms, we set the initial values of leaves as $Q_{\text{init}}(s, a) = \log \pi_{\text{init}}(a|s)$, with $\pi_{\text{init}}(a|s) \propto \exp(\hat{Q}(s, a)/\tau_{\text{init}})$, \hat{Q} denoting the Q-network and $\tau_{\text{init}} = 0.1$, same as the authors. In ANTS, we just set $Q_{\text{init}} = \hat{Q}$. For reference, we also provide the scores of DQN using our pretrained networks. In all cases, we report the mean of 24 planning runs, with a random number

Table 1. Results on Atari using Q-networks pretrained with DQN.

	DQN	MENTS	TENTS	ANTS
ALIEN	1348.8	1169.2	1554.6	2898.3
AMIDAR	175.4	19.3	108.9	173.0
ASTERIX	2241.7	20358.3	54095.8	49483.3
ASTEROIDS	731.7	3602.5	4550.4	5149.6
ATLANTIS	159841.7	224400.0	248058.3	199729.2
BANK HEIST	591.3	46.7	105.4	687.5
BEAM RIDER	6876.1	1448.8	2123.0	10807.8
BREAKOUT	218.4	29.8	10.0	356.0
CENTIPEDE	1283.0	115442.0	61625.1	33476.0
DEMON ATTACK	3142.5	20920.0	26342.3	70401.9
ENDURO	553.7	200.0	199.2	651.5
FROSTBITE	156.7	129.6	136.3	183.8
GOPHER	4288.3	8550.0	8715.0	10209.2
HERO	20612.1	2235.0	9126.7	20599.0
MS PACMAN	2878.8	2902.5	3578.8	5289.2
PHOENIX	5060.4	5209.6	5219.0	8493.3
QBERT	9691.7	13254.1	14653.1	15308.3
ROBOTANK	32.8	4.7	3.8	38.0
SEAQUEST	1322.5	826.7	523.3	2789.2
SOLARIS	216.6	1236.7	734.2	233.3
SPACE INVADERS	2022.3	708.3	887.7	4746.9
WIZARD OF WOR	1595.8	6241.7	11654.2	13450.0

of no-op actions from the $[0, 30]$ interval at the beginning of each episode. The results are shown in Table 1.

As we can see, ANTS outperforms all baselines in 16 out of 22 tasks. In some cases, the difference of scores is very large - for instance in Alien, Demon Attack and Seaquest. We see this as a proof of the importance of temperature adaptation in maximum entropy planning approaches. We can also see a perhaps surprising result: in some tasks, the performance of MENTS and TENTS is actually *worse* than that of the Q-networks those algorithms use - for instance in Amidar, Bank Heist and Beam Rider. This phenomenon, ostensibly contradicting the results of Xiao et al. (2019) and Dam et al. (2020), may be explained by several factors:

1. The results we report are using the same hyperparameters across all tasks, including the sensitive temperature parameter, that depends on the scale of the rewards. The authors of MENTS and TENTS tune this parameter on each task separately, which causes their reported results to be relatively higher.
2. In the experimental setup of MENTS, the planner is run only once every 10 steps. In the remaining steps, DQN is used for action selection - see Appendix A in Xiao et al. (2019). We do not know if the authors of TENTS follow the same practice, but it could hide some potential deficiencies of those two algorithms.
3. We use different Q-network checkpoints for leaf evaluation, so some differences may be caused by the vari-

ance of DQN training.

5.2. Ablations

In order to measure the contribution of the different components of ANTS, we perform additional experiments on Asteroids and Qbert, removing the individual features of our method. We compare the performance of full ANTS with two other variants:

- ANTS-SQL, using the soft Q-iteration backup (3) instead of soft policy iteration (5), and
- ANTS-SAMPLE, sampling from the target distribution π_τ instead of greedy action selection.

We also provide the MENTS scores for reference. The results are shown in Figure 1.

On Asteroids, we can see clearly that each part of the algorithm improves the final performance. On Qbert, the results are less conclusive - using the soft Q-iteration backup does degrade the scores to a small degree, but the performance of ANTS with action sampling is very close to full ANTS. In both cases, each variant achieves significantly higher scores than MENTS.

5.3. Closed-Loop Training

In our final set experiment, we measure the applicability of ANTS for end-to-end training starting from a randomly-initialized network, in a similar fashion to Silver et al.

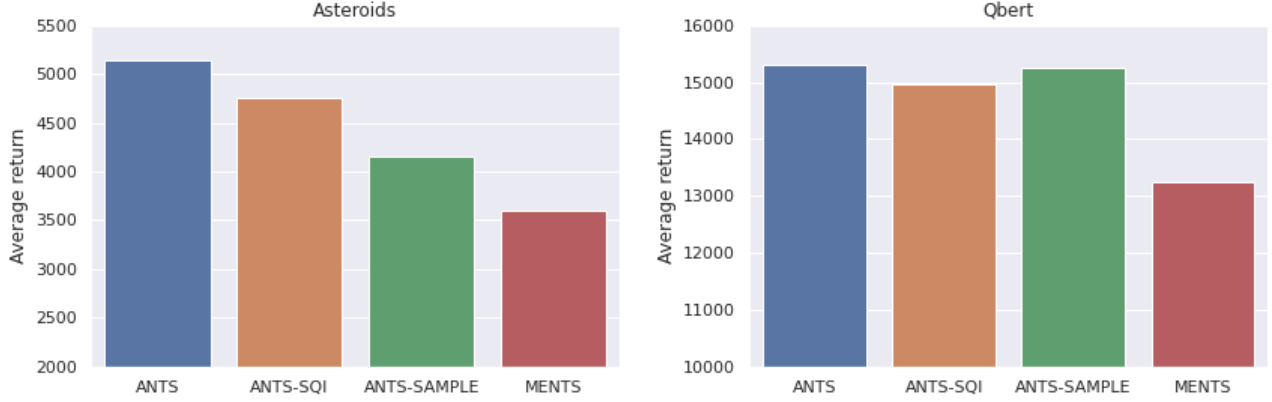


Figure 1. Ablations of ANTS on Asteroids and Qbert.

Table 2. Results on Atari using Q-networks pretrained with DQN. ANTS-CL denotes the ANTS agent with closed-loop training. We report a median of 3 runs.

	DQN	ANTS	ANTS-CL
ALIEN	1348.8	2898.3	958.3
ASTERIX	2241.7	49483.3	50629.2
ASTEROIDS	731.7	5149.6	2530.4
BANK HEIST	591.3	687.5	466.7
FROSTBITE	156.7	183.8	252.5

(2018). This is a feature that, to the best of our knowledge, has not been demonstrated yet in maximum entropy MCTS methods. For this purpose, we run ANTS in a closed loop with Q-network training, using the Q-values estimated by the planner as learning targets.

For this experiment, we make several adjustments to ANTS. First, we modify the network architecture to inject information about the current temperature - the learning targets are influenced by this value so, as we have discovered, giving the network access to this information greatly stabilizes the algorithm. Second, in order to provide diverse data for training, instead of selecting actions that maximize the estimated Q-values, we sample from the softmax distribution defined by the Q-values with temperature 0.5. Third, to make good use of the computational resources, we limit the number of MCTS passes per action to 30. Finally, we decrease \mathcal{H}_{\min} to 0.1. We show the results on 5 tasks in Table 2. We provide the scores of DQN and ANTS with the DQN network for reference. However, we note that those results are not directly comparable - the DQN agent consumes 200M frames for training, while our ANTS closed-loop agent consumes 5-10M, depending on the task¹. On the other hand, ANTS

closed-loop has access to a model of the MDP, which DQN does not use.

As we can see, our closed-loop agent achieves respectable performance even with just 30 planning passes. Especially the Asterix results show the capability of ANTS to replicate the performance of a planner with a high computational budget of 500 passes using a modest budget of 30 passes and an appropriately trained neural network.

6. Conclusions

In this work, we have presented Adaptive Entropy Tree Search: a robust and efficient approach to planning, based on combining temperature adaptation with maximum entropy Monte Carlo planning. We have demonstrated its ability to leverage pre-trained Q-networks to guide and improve planning, as well as the capacity to train Q-networks in a closed loop. We identify several promising directions for future research. First, it would be beneficial to study the interplay between planning and learning in the context of maximum entropy methods - it is believed that those approaches aid in smoothing the optimization landscape of neural networks. Second, a more extensive empirical evaluation of the components of ANTS may reveal ideas for further improvements. Third, supplementing ANTS with learning a model of the MDP and increasing the computational budget of the closed-loop experiments would enable comparison with MuZero, to evaluate the benefits of maximum entropy methods in large-scale planning systems. We believe that such research would serve as a step towards developing practical planning approaches to challenging real-world problems.

of the experiment - 3 days with 24 parallel environments on a machine with two Intel Xeon E5-2697 v3 CPUs.

¹The amount of experience used was limited by the runtime

References

- Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016.
- Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the Atari human benchmark. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 507–517. PMLR, 13–18 Jul 2020. URL <http://proceedings.mlr.press/v119/badia20a.html>.
- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, Vol. 47:253–279, 2012. doi: 10.1613/jair.3912. URL <http://arxiv.org/abs/1207.4708>. cite arxiv:1207.4708.
- Brent, R. *Algorithms for minimization without derivatives*. Prentice-Hall, 1973. ISBN 0-13-022335-2.
- Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez Liebana, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1:1–43, 03 2012. doi: 10.1109/TCIAIG.2012.2186810.
- Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL <http://arxiv.org/abs/1812.06110>.
- Chaslot, G., Jong, S., Saito, J.-T., and Uiterwijk, J. Monte-carlo tree search in production management problems. *Belgian/Netherlands Artificial Intelligence Conference*, 01 2006.
- Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018. URL <http://arxiv.org/abs/1805.12114>.
- Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. Model-based reinforcement learning via meta-policy optimization. *CoRR*, abs/1809.05214, 2018. URL <http://arxiv.org/abs/1809.05214>.
- Dam, T., D’Eramo, C., Peters, J., and Pajarinen, J. Convex regularization in monte-carlo tree search, 2020.
- Eysenbach, B. and Levine, S. If maxent rl is the answer, what is the question?, 2019.
- Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., and Munos, R. Monte-Carlo tree search as regularized policy optimization. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3769–3778. PMLR, 13–18 Jul 2020. URL <http://proceedings.mlr.press/v119/grill20a.html>.
- Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31, pp. 2450–2462. Curran Associates, Inc., 2018. URL <https://proceedings.neurips.cc/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf>.
- Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1352–1361, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL <http://proceedings.mlr.press/v70/haarnoja17a.html>.
- Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/haarnoja18b.html>.
- Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications, 2019.
- Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., and Dragan, A. D. Inverse reward design. *CoRR*, abs/1711.02827, 2017. URL <http://arxiv.org/abs/1711.02827>.
- Hafner, D., Lillicrap, T. P., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *CoRR*, abs/1912.01603, 2019. URL <http://arxiv.org/abs/1912.01603>.
- Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Pfaff, T., Weber, T., Buesing, L., and Battaglia, P. W. Combining q-learning and search with amortized value esti-

- mates. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020a. URL <https://openreview.net/forum?id=SkeAaJrKDS>.
- Hamrick, J. B., Friesen, A. L., Behbahani, F., Guez, A., Viola, F., Witherspoon, S., Anthony, T., Buesing, L., Veličković, P., and Weber, T. On the role of planning in model-based deep reinforcement learning, 2020b.
- Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. 2018. URL <https://openreview.net/forum?id=rk07ZXZRb>.
- Heess, N., Wayne, G., Silver, D., Lillicrap, T. P., Tassa, Y., and Erez, T. Learning continuous control policies by stochastic value gradients. *CoRR*, abs/1510.09142, 2015. URL <http://arxiv.org/abs/1510.09142>.
- Hernando, A., Hernando, R., Plastino, A., and Plastino, A. R. The workings of the maximum entropy principle in collective human behaviour. *Journal of The Royal Society Interface*, 10(78):20120758, Jan 2013. ISSN 1742-5662. doi: 10.1098/rsif.2012.0758. URL <http://dx.doi.org/10.1098/rsif.2012.0758>.
- Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In McIlraith, S. A. and Weinberger, K. Q. (eds.), *AAAI*, pp. 3215–3222. AAAI Press, 2018. URL <http://dblp.uni-trier.de/db/conf/aaai/aaai2018.html#HesselMHSODHPAS18>.
- Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., and Michalewski, H. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019. URL <http://arxiv.org/abs/1903.00374>.
- Khandelwal, P., Liebman, E., Niekum, S., and Stone, P. On the analysis of complex backup strategies in monte carlo tree search. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1319–1328, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <http://proceedings.mlr.press/v48/khandelwal16.html>.
- Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014. URL <http://arxiv.org/abs/1412.6980>. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
- Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pp. 282–293. Springer, 2006.
- Kozakowski, P., Januszewski, P., Czechowski, K., Kuciński, Ł., and Miłoś, P. Structure and randomness in planning and reinforcement learning. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. URL https://openreview.net/forum?id=x1X_kcgVxP3.
- Lee, C.-S., Wang, M.-H., Chaslot, G., Hoock, J.-B., Rimmel, A., Teytaud, O., Tsai, S.-R., Hsu, S.-C., and Hong, T.-P. The computational intelligence of mogo revealed in taiwan’s computer go tournaments. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1: 73 – 89, 04 2009. doi: 10.1109/TCIAIG.2009.2018703.
- Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018.
- Mahlmann, T., Togelius, J., and Yannakakis, G. Towards procedural strategy game generation: Evolving complementary unit types. volume 6624, pp. 93–102, 04 2011. ISBN 978-3-642-20524-8. doi: 10.1007/978-3-642-20525-5_10.
- Mansley, C., Weinstein, A., and Littman, M. Sample-based planning for continuous action markov decision processes. 01 2011.
- Miłoś, P., Kuciński, Ł., Czechowski, K., Kozakowski, P., and Klimek, M. Uncertainty-sensitive learning and planning with ensembles. *arXiv preprint arXiv:1912.09996*, 2019.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 00280836. URL <http://dx.doi.org/10.1038/nature14236>.
- Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Bridging the gap between value and policy based reinforcement learning. *CoRR*, abs/1702.08892, 2017. URL <http://arxiv.org/abs/1702.08892>.
- Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T. (eds.), *ICML*, pp. 807–814. Omnipress, 2010. URL <http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10>.
- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. Mastering

- atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL <http://arxiv.org/abs/1911.08265>.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 1144: 1140–1144, 2018.
- Sutton, R. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 08 1988. doi: 10.1007/BF00115009.
- Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4): 160–163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL <https://doi.org/10.1145/122344.122377>.
- Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.
- Tanabe, Y., Yoshizoe, K., and Imai, H. A study on security evaluation methodology for image-based biometrics authentication systems. In *2009 IEEE 3rd International Conference on Biometrics: Theory, Applications, and Systems*, pp. 1–6, 2009. doi: 10.1109/BTAS.2009.5339016.
- Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba, J. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019. URL <http://arxiv.org/abs/1907.02057>.
- Xiao, C., Huang, R., Mei, J., Schuurmans, D., and Müller, M. Maximum entropy monte-carlo planning. In *Advances in Neural Information Processing Systems*, volume 32, pp. 9520–9528. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/7ffb4e0ece07869880d51662a2234143-Paper.pdf>.
- Ziebart, B. D. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.
- Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In Fox, D. and Gomes, C. P. (eds.), *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 1433–1438. AAAI Press, 2008. URL <http://www.aaai.org/Library/AAAI/2008/aaai08-227.php>.

A. Appendix

A.1. Algorithm

The pseudocode of the ANTS algorithm is shown in [Algorithm 1](#). The function ANTS is called at every step of the episode to determine the action to take. At the beginning, the temperature $\tilde{\tau}$ is set to τ_0 . Every *adaptation_frequency* passes it is adjusted, and the Q-values in the tree are recalculated based on its new value. The temperature is transferred between episode steps. To improve efficiency, we also transfer the planning tree between steps - both for ANTS and for all baselines.

Algorithm 1 Adaptive Entropy Tree Search.

```

function ANTS(root, model)
  for i in  $\{1 \dots n\_passes\}$  do
    (leaf, rewards)  $\leftarrow$  SELECT(root, model,  $\tilde{\tau}$ )
    leaf.children  $\leftarrow$  EXPAND(leaf, model)
     $\forall_a \text{leaf.children}[a].qvalue$ 
       $\leftarrow$  SIMULATE(leaf.state, a,  $\tilde{\tau}$ )
    BACKPROPAGATE(leaf, rewards,  $\tilde{\tau}$ )
    if i mod adaptation_frequency = 0 then
       $\tau \leftarrow$  ADAPT_TEMPERATURE(root)
       $\tilde{\tau} \leftarrow \exp(\alpha \log \tilde{\tau} + (1 - \alpha) \log \tau)$ 
      RECALCULATE_QVALUES(root,  $\tilde{\tau}$ )
    end if
  end for
  return FINAL_ACTION(root,  $\tilde{\tau}$ )
end function
    
```

Temperature adaptation is performed by minimizing the objective function defined in equation (8), computed over the internal nodes, as shown in [Algorithm 2](#). Q-values are recalculated according to (10) applied to the nodes started from the leaves, using depth-first search.

Algorithm 2 Temperature adaptation.

```

function ADAPT_TEMPERATURE(root)
  function LOSS( $\tau$ )
    nodes  $\leftarrow$  INTERNAL_NODES(root)
     $\forall_{n \in \text{nodes}} \mathcal{H}_n \leftarrow \mathcal{H}(\text{SOFTMAX\_POLICY}(n, \tau))$ 
     $\forall_{n \in \text{nodes}} d_n \leftarrow \max(\mathcal{H}_{\min} - \mathcal{H}_n, \mathcal{H}_n - \mathcal{H}_{\max}, 0)$ 
    return  $\left( \frac{1}{|\text{nodes}|} \sum_{n \in \text{nodes}} d_n \right) + \beta \log \tau$ 
  end function
  return MINIMIZE(LOSS)
end function

function RECALCULATE_QVALUES(node,  $\tau$ )
  for child in node.children do
    RECALCULATE_QVALUES(child,  $\tau$ )
  end for
  node.qvalue  $\leftarrow$  QVALUE(node,  $\tau$ )
end function
    
```

The selection phase is shown in [Algorithm 3](#). It selects actions inside the tree according to equation (11). We also limit the length of the path using a hyperparameter *depth_limit*. This is omitted in the pseudocode for clarity.

Algorithm 3 Selection phase.

```

function SELECT(root, model,  $\tau$ )
  node  $\leftarrow$  root
  state  $\leftarrow$  root.state
  rewards  $\leftarrow$  ()
  while not IS_LEAF(node) do
    action  $\leftarrow$  TREE_ACTION(node,  $\tau$ )
    (state, reward)  $\leftarrow$  model(state, action)
    rewards  $\leftarrow$  rewards  $\oplus$  (reward)
    node  $\leftarrow$  node.children[action]
  end while
  return (node, rewards)
end function

function TREE_ACTION(node,  $\tau$ )
   $\pi_{\text{soft}} \leftarrow$  SOFTMAX_POLICY(node,  $\tau$ )
   $\forall_a \pi_{\text{emp}}(a) \leftarrow \text{node.children}[a].count / \text{node.count}$ 
  return  $\text{argmax}_a [\pi_{\text{soft}}(a) - \pi_{\text{emp}}(a)]$ 
end function
    
```

We omit the expansion phase for brevity. In the simulation phase, we just run the Q-network.

```

function SIMULATE(state, action,  $\tau$ )
  return  $\hat{Q}(\text{state}, \text{action}, \tau)$ 
end function
    
```

In the backpropagation phase, we traverse the selected path bottom-up and update the Q-values.

```

function BACKPROPAGATE(node, rewards,  $\tau$ )
  for reward in REVERSED(rewards) do
    node.reward  $\leftarrow$  reward
    node.count  $\leftarrow$  node.count + 1
    node.qvalue  $\leftarrow$  QVALUE(node,  $\tau$ )
    node  $\leftarrow$  node.parent
  end for
end function
    
```

We sample the final action from the softmax policy, rescaled by the action selection temperature τ_a . In the pretrained Q-network experiments, we set τ_a to a very low value, so this is equivalent to argmax with random tie breaking.

```

function FINAL_ACTION(node,  $\tau$ )
  sample a  $\sim$  SOFTMAX_POLICY(node,  $\tau \cdot \tau_a$ )
  return a
end function
    
```

We provide the functions for calculating the softmax policy and the Q-values in [Algorithm 4](#).

Algorithm 4 Auxiliary functions.

```

function SOFTMAX_POLICY(node,  $\tau$ )
   $\pi(a) \propto \exp(\frac{1}{\tau} \text{node.children}[a].qvalue)$ 
  return  $\pi$ 
end function

function QVALUE(node,  $\tau$ )
  if node.children = () then
    return node.qvalue
  end if
   $\pi \leftarrow \text{SOFTMAX\_POLICY}(\text{node}, \tau)$ 
   $\forall_a pr_a \leftarrow -\tau \log \pi(a) - \tau \log |\mathcal{A}|$ 
   $v \leftarrow \sum_a \pi(a)(\text{node.children}[a].qvalue + pr_a)$ 
  return node.reward +  $\gamma v$ 
end function

```

A.2. Complexity Analysis

We first consider a variant of the algorithm without transferring the tree between steps. We denote $n = \text{n_passes}$, $m = \text{n_passes/adaptation_frequency}$, $d = \text{depth_limit}$ and $A = |\mathcal{A}|$. In each call of ANTS we perform n planning passes. In each pass, we traverse a path of length $O(d)$, choosing an action out of A in each node. This gives us a time complexity of $O(ndA)$ for running the MCTS phases. Additionally, in m of the passes we perform temperature adaptation. We assume that the optimization only requires a constant number of steps. In each step, we compute the entropies of all internal nodes in the tree. There is n of them - each has been expanded in some previous planning pass. Computing the entropy requires calculating a softmax over A actions. Hence, the time complexity of temperature adaptation is $O(mnA)$, and the time complexity of the whole algorithm is $O(nA(d + m))$. We only need to store the tree in memory. Each node stores links to its A children, and we have n internal nodes, so the space complexity is $O(nA)$.

In the case of keeping the tree between episode steps, the tree can grow up to $O(nd)$ nodes - in each pass we expand n nodes, and we can only "see" nodes expanded in the last d steps. All other considerations stay the same as in the previous variant, which yields a time complexity of $O(ndA)$ for the MCTS phases and $O(nmdA)$ for temperature adaptation, so $O(nmdA)$ in total, and a space complexity of $O(ndA)$. However, note that keeping the tree is an optimization, achieved by reusing the computation performed in the previous episode steps. As such, when keeping the tree, the number of passes can be reduced to match the performance of the variant without keeping the tree, and then the final runtime should be lower.

A.3. Hyperparameters

In [Table 3](#) we list the hyperparameter values, together with ranges considered during tuning. Tuning was performed on a subset of three games: Amidar, Ms Pacman and Qbert, in small grid searches of 2-3 hyperparameters at a time.

In the pretrained Q-network experiments, we use same network architecture as in (Mnih et al., 2015). In the closed-loop training experiments, we use smaller networks, of the following architecture:

1. Five convolutional layers of depth 16, kernel size 3×3 and stride 2×2 .
2. Dense hidden layer of depth 64.
3. Temperature injection (see below).
4. Dense layer predicting the action Q-values.

For the temperature injection, we take a logarithm of the temperature, apply a linear layer of depth 64 and merge the two inputs using layer normalization (Ba et al., 2016) followed with an element-wise sum.

All activations are rectified linear units (Nair & Hinton, 2010). We train the networks using the Adam optimization algorithm (Kingma & Ba, 2014) with learning rate 0.001.

A.4. Code

The code and hyperparameter settings used to run all experiments in this work is available open-source at <https://github.com/adaptive-entropy-tree-search/ants>.

A.5. Experimental Details

We perform our experiments on 22 Atari games from the [Arcade Learning Environment](#). We apply the following preprocessing to those environments:

- Repeating each action for 4 consecutive steps, taking a maximum of 2 last frames as the observation.
- Stacking 4 last frames obtained from the previous step in one observation.
- Gray-scale observations, cropped and rescaled to size 84×84 .
- Random number of no-op actions from range $[0..30]$ at the beginning of each episode.
- Time limit of 10K interactions with the environment.

Robust and Efficient Planning using Adaptive Entropy Tree Search

Hyperparameter	Value	Considered values
\mathcal{H}_{\min} - minimum entropy (pretrained Q-network)	0.5	{0.1, 0.3, 0.5, 0.7, 1.0}
\mathcal{H}_{\min} - minimum entropy (closed-loop training)	0.1	{0.1, 0.3, 0.5, 0.7, 1.0}
\mathcal{H}_{\max} - maximum entropy	1.0	{0.7, 1.0, 1.5, 1.8}
τ_0 - initial temperature	1.0	{1.0}
α - temperature EMA decay	0.9	{0.0, 0.8, 0.9, 0.95}
β - regularization coefficient	0.001	{0.001}
γ - discount factor	0.99	{0.99}
τ_a - action selection temperature (pretrained Q-network)	0.001	{0.001}
τ_a - action selection temperature (closed-loop training)	0.5	{0.1, 0.3, 0.5, 1.0}
n_passes (pretrained Q-network)	500	{500}
n_passes (closed-loop training)	30	{10, 20, 30, 50}
depth_limit	50	{10, 20, 30, 50, 100}
adaptation_frequency	Equal to n_passes.	

Table 3. Hyperparameter values and considered ranges.

We do not use reward clipping in any of the experiments.

Each experiment has been run on a single machine with two Intel Xeon E5-2697 v3 CPUs. The pretrained Q-network experiments took at most 2 days each, and the closed-loop experiments were terminated after 3 days.