Planning and Policy Improvement

Ivo Danihelka

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Computer Science
University College London

March 12, 2023

I, Ivo Danihelka, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

MuZero is currently the most successful general reinforcement learning algorithm, achieving the state of the art on Go, chess, shogi, and Atari. We want to help MuZero to be successful in even more domains. Towards that, we do three steps: 1) We identify MuZero's problems on stochastic environments and provide ways to model enough information to support causally correct planning. 2) We develop a strong baseline agent on Atari. This agent, named Muesli, matches the state of the art on Atari, even without deep search. The conducted ablations inform us about the importance of model learning, deep search, large networks, and regularized policy optimization. 3) Because MuZero's tree search is very helpful on Go and chess, we use the principle of policy improvement to design search algorithms with even better properties. The new algorithms, named Gumbel AlphaZero and Gumbel MuZero, match the state of the art on Go, chess, and Atari, and significantly improve prior performance when planning with few simulations.

Impact Statement

If you can measure something, you can optimize it. By improving reinforcement learning algorithms, we improve the science of sequential decision making. The possible practical applications are limited mainly by our ability to express our wishes. We can optimize the win rate in board games, the score in computer games, the path length in transportation, the bandwidth and latency in communication, the efficiency in industrial process control, the log-probability in video compression, and lane following in self-driving cars.

The provided Gumbel AlphaZero and Gumbel MuZero guarantee a policy improvement if the visited actions are correctly evaluated. This increases our confidence that these algorithms will work on new domains, even on domains with a large number of actions.

Acknowledgements

I want to thank many people:

Matteo Hessel for being the ideal collaborator: smart, fast, vigilant, and supportive.
David Silver for his interest, advice, and willingness to help.
Peter Dayan for sharing his wisdom and detailed comments.
Fabio Viola for our long continuing collaboration.
Igor Babuschkin for driving the AlphaStar development.
Manuel Kroiss, *Iurii Kemaev*, *Dan Horgan*, *Charlie Beattie* and *Fan Yang* for heroic development and user support.
Joseph Modayil and *Loic Matthey* for very helpful comments on our paper drafts.
Danilo Rezende for doing inspirational research.
Daan Wierstra for being a great mentor.
Julian Schrittwieser, *Thomas Hubert*, and *Ioannis Antonoglou* for their help and support with MuZero.
George Papamakarios, Ray Jiang, Nan Rosemary Ke, Theophane Weber, Karol Gregor, Hamza Merzic, Jane Wang, Jovana Mitrovic, Frederic Besse, Lars Buesing, Arthur Guez, Simon Schmitt, Laurent Sifre, and *Hado van Hasselt* for collaborating with me.
JAX (Bradbury et al., 2018) core developers for making excellent design choices.
Andreas Fidjeland, *Arun Nair*, *Koray Kavukcuoglu*, *Charles Blundell*, *Alexander Graves*, *Gregory Wayne*, *Alexander Pritzel*, *Benigno Uria*, *Jonathan Hunt*, *Surya Bhupatiraju*, and other friends at DeepMind for their friendship.

This thesis is dedicated to all who help others to do more.

Research Paper Declaration Form

Published paper: *Causally correct partial models for reinforcement learning*

Where was the work published? *arXiv*

Who published the work? *arXiv*

When was the work published? *2020*

Was the work subject to academic peer review? *No*

Have you retained the copyright for the work? *Yes*

A statement of contribution covering all authors:

– *Danilo Rezende* explained the problem in the notation of causality, proposed the usage of the backdoor adjustments, invented conditioning on the intended action, cowrote the paper.

– *Ivo Danihelka* initialized the research, identified the MuZero problems on stochastic environments, proposed solutions, implemented the algorithms by modifying MuZero to support Monte-Carlo tree search with chance nodes, ran experiments on MiniPacman.

– *George Papamakarios* coauthoed the search algorithms, ran experiments on MiniPacman, prepared MDP diagrams, cowrote the paper.

– *Nan Rosemary Ke* prepared 3D experiments.

– *Ray Jiang* prepared the analytical value-iteration analysis on MDPs.

– *Theophane Weber* helped with the paper writing.

– *Karol Gregor* advised the algorithm design, prepared Dyna experiments.

– *Hamza Merzic* prepared training scripts for 3D environments.

– *Fabio Viola* prepared 3D environments.

– *Jane Wang* ran 3D experiments.

– *Jovana Mitrovic* investigated causality, assisted with 3D experiments.

– *Frederic Besse* prepared 3D environments.

– *Ioannis Antonoglou* implemented a Python MuZero, reviewed code changes.

– *Lars Buesing* promoted causality research.

In which chapter(s) of your thesis can this material be found? Chapter 3

Published paper: *Muesli: combining improvements in policy optimization*

Where was the work published? *Proceedings of the 38th International Conference on Machine Learning*

Who published the work? *Proceedings of Machine Learning Research*

When was the work published? *2021*

Was the work subject to academic peer review? *Yes*

Have you retained the copyright for the work? *Yes*

A statement of contribution covering all authors:

– *Matteo Hessel* developed the fast and reliable training framework, coauthored a MuZero implementation, ran experiments, wrote the final version of the paper.

– *Ivo Danihelka* designed and implemented the Muesli algorithms, ran experiments, prepared the plots, wrote the first version of the paper.

– *Fabio Viola* coauthored a MuZero implementation, maintained the training framework.

– *Arthur Guez* helped with the MuZero implementation, ran experiments on Go.

– *Simon Schmitt* implemented Reanalyze, ran experiments on continuous control.

– *Laurent Sifre* prepared a fast batched Monte-Carlo tree search, designed the fast prediction and dynamics network.

– *Theophane Weber* motivated the research.

– *David Silver* organized a nice team of people, advised the experiments.

– *Hado van Hasselt* kept finding counterexamples.

In which chapter(s) of your thesis can this material be found? Chapter 4

Published paper: *Policy improvement by planning with Gumbel*

Where was the work published? *Proceedings of the International Conference on Learning Representations*

Who published the work? *International Conference on Learning Representations*

When was the work published? *2022*

Was the work subject to academic peer review? *Yes*

Have you retained the copyright for the work? *Yes*

A statement of contribution covering all authors:

– *Ivo Danihelka* designed and implemented the algorithms, ran experiments, prepared the paper, prepared the open-source release.

– *Arthur Guez* prepared Python training on Go, reviewed the paper draft.

– *Julian Schrittwieser* helped with the usage of the AlphaZero and MuZero codebase, reviewed the Gumbel MuZero C++ changes, provided the faster AlphaZero network architecture.

– *David Silver* advised the algorithm design, provided a historical background, rewrote the introduction.

In which chapter(s) of your thesis can this material be found? Chapter 5

e-Signatures confirming that the information above is accurate:

Candidate: *Ivo Danihelka* Date: *15 August 2022*

Supervisor: *David Silver* Date: *12 September 2022*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Monte-Carlo tree search (MCTS) revolutionized computer Go and contributed to the stunning victories of AlphaGo (Silver et al., 2016). Subsequently, AlphaZero (Silver et al., 2018) brought these successes to chess and shogi. These search methods use a resettable simulator, which allows us to simulate multiple actions from a state (e.g., by saving the game state).

However, resettable simulators are not available for many environments. For example, we cannot reset the state of the real world. Therefore, Schrittwieser et al. (2020) developed MuZero, which learns a model of the environment.

In Chapter 3, we point out that the MuZero model was developed for deterministic environments. We show the possible problems on stochastic environments and provide a spectrum of causally correct planning methods. The same problems also impact Value Prediction Networks (Oh et al., 2017), Dreamer (Hafner et al., 2020), and similar variants of Dyna (Sutton, 1990) with multi-step sampled rollouts.

After that we develop two strong algorithms. The first algorithm, called Muesli, matches the state of the art on Atari, even without a deep search (Chapter 4). Next, we obtain great performance also on Go and chess with the second algorithm, called Gumbel MuZero. Compared to MuZero, the new Gumbel MuZero supports stochastic policies and keeps improving the policy even with a small number of simulations (Chapter 5). By being principled, the produced algorithms remove problem-dependent hyperparameters and are applicable also to Expert Iteration (Anthony et al., 2017).

1.1 Thesis outline

Chapter 2 provides the background on Monte-Carlo tree search, AlphaZero, and MuZero.

Chapter 3 warns against incorrect planning with the MuZero model on stochastic environments. We then propose a family of models for causally correct planning. The chapter is based on:

- Danilo J. Rezende*, Ivo Danihelka*, George Papamakarios, Nan Rosemary Ke, Ray Jiang, Theophane Weber, Karol Gregor, Hamza Merzic, Fabio Viola, Jane Wang, Jovana Mitrovic, Frederic Besse, Ioannis Antonoglou, Lars Buesing. *Causally correct partial models for reinforcement learning* (Rezende et al., 2020).

Chapter 4 proposes a policy update which does not require a deep search, yet matches the MuZero state-of-the-art results on Atari. The chapter is based on:

- Matteo Hessel*, Ivo Danihelka*, Fabio Viola, Arthur Guez, Simon Schmitt, Laurent Sifre, Theophane Weber, David Silver, Hado van Hasselt. *Muesli: combining improvements in policy optimization* (Hessel et al., 2021).

Chapter 5 proposes a policy improvement algorithm based on sampling without replacement and uses the algorithm to design a principled AlphaZero tree search. The chapter is based on:

- Ivo Danihelka, Arthur Guez, Julian Schrittwieser, David Silver. *Policy improvement by planning with Gumbel* (Danihelka et al., 2022).

The above papers are a result of collaboration with excellent coauthors. Especially, I want to give credit to Danilo Rezende for introducing me to causality and for proposing conditioning on the intended action. When looking at the second paper, I want to thank Matteo Hessel for developing the underlying distributed training framework, coauthoring a MuZero implementation, running many experiments, writing the final version of the paper, and reviewing my code rapidly. Finally, I'm grateful for wise steering from David Silver; we avoided many wrong paths. Now

I will declare my contributions. On the above papers, I am the joint first author. I initialized the research, proposed and implemented the algorithms, ran the majority of the experiments, and produced the analysis. Moreover, in the second and third papers, I authored all theorems and proofs.

We have published also the following papers, which are beyond the scope of this thesis:

- *Comparison of maximum likelihood and GAN-based training of Real NVPs* (Danihelka et al., 2017).
- *The Cramer distance as a solution to biased Wasserstein gradients* (Bellemare et al., 2017).
- *Video pixel networks* (Kalchbrenner et al., 2017).
- *Neural scene representation and rendering* (Eslami et al., 2018).
- *OpenSpiel: A framework for reinforcement learning in games* (Lanctot et al., 2019).
- *Grandmaster level in StarCraft II using multi-agent reinforcement learning* (Vinyals et al., 2019).
- *Learning by directional gradient descent* (Silver et al., 2022).

Chapter 2

Common Background

We will describe the reinforcement learning setting, the used notation, and the tree search with AlphaZero and MuZero. The ideal reader would be already aware of AlphaZero (Silver et al., 2018) or MuZero (Schrittwieser et al., 2020) and wants to learn how to improve them.

For an introduction to reinforcement learning, we strongly recommend the freely-available book by Sutton and Barto (2018). Along with many other useful concepts, the book explains bandits, Markov decision processes, episodes, multi-step returns, dynamic programming, and policy gradients.

2.1 The environment and the objective

We are interested in episodic environments with variable episode lengths (e.g., Atari games), formalized as Markov Decision Processes (MDPs) with an initial state distribution μ and a discount $\gamma \in [0,1]$; ends of episodes correspond to absorbing states with no rewards.

2.1.1 The objective

The agent starts at a state $S_0 \sim \mu$ from the initial state distribution. At each time step t, the agent takes an action $A_t \sim \pi(A_t|S_t)$ from a *policy* π, obtains the reward R_{t+1} and transitions to the next state S_{t+1}. The expected sum of discounted rewards

after a state-action pair is called the *action-value* or *Q-value* $q_\pi(s,a)$:

$$q_\pi(s,a) = \mathbb{E}\left[\sum_{t=0} \gamma^t R_{t+1} | \pi, S_0 = s, A_0 = a\right], \tag{2.1}$$

where the sum of rewards $\sum_{t=0} \gamma^t R_{t+1}$ is a *return*. The *value* of a state s is $v_\pi(s) = \mathbb{E}_{A\sim\pi(\cdot|s)}[q_\pi(s,A)]$ and the objective is to find a policy π that maximizes the expected value of the states from the initial state distribution:

$$J(\pi) = \mathbb{E}_{S\sim\mu}[v_\pi(S)]. \tag{2.2}$$

For example, if the reward is the increment to the score in a game, the objective is to maximize the expected score. When evaluating an agent, we can estimate the expected score by the average score over many episodes.

2.2 Learning and policy improvement

The objective (Eq. 2.2) can be directly maximized by evolution strategies (Wierstra et al., 2014; Salimans et al., 2017) or by policy gradients (Williams, 1992; Sutton et al., 2000; Agarwal et al., 2019; Agarwal et al., 2020a). These methods estimate the gradient of the objective $J(\pi)$ with respect to the policy parameters and are therefore compatible with stochastic gradient descent and function approximation. In the purest form, the unbiased estimate of the gradient is obtained from on-policy data and the policy is updated only after the end of an episode.

We will instead focus on improving the value at states visited by a possibly older policy π_{prior}. We will use a policy improvement algorithm for that. Given a state s, a *policy improvement* algorithm constructs a new policy π' such that $v_{\pi'}(s) \geq v_{\pi_{\text{prior}}}(s)$. For instance, given the π_{prior} and its Q-values $q_{\pi_{\text{prior}}}(s,a)$, the basic policy improvement algorithm constructs the greedy policy:

$$\arg\max_{\pi'} \mathbb{E}_{A\sim\pi'(\cdot|s)}\left[q_{\pi_{\text{prior}}}(s,A)\right]. \tag{2.3}$$

After producing the improved policy at a state, we can update the agent's policy

π to be closer to the improved policy π'. Because the agent's policy is usually represented by a function approximator (e.g., a policy network), we must be careful not to degrade the value at other important states. For the careful policy update, we will delve into regularized policy optimization in Chapter 4.

2.3 Policy improvement by search

Because the Q-values are usually unknown, we need to estimate them. Let us assume that we have a simulator or a learned model of the environment. We can then estimate the Q-values by a tree search.

2.3.1 Monte-Carlo tree search

Monte-Carlo tree search (MCTS) is a family of algorithms for finding the optimal action at a state. To estimate the Q-value $q_\pi(s,a)$ of an action a, we can simulate a trajectory starting from the state s, taking the action a, and then following the policy π. Such on-policy simulation is called a *rollout* and the obtained sum of rewards (aka *Monte-Carlo return*) is an estimate of the Q-value.

The Monte-Carlo tree search adds one more idea: to select the optimal action, we need the optimal Q-values $q^*(s,a)$ of the optimal policy π^*, instead of the Q-values $q_\pi(s,a)$ of a policy π. This means we should search for optimal actions also at the states visited inside a simulation.

There are many ways to find the optimal action. For a small environment, we can construct the whole *game tree* with states at the nodes and actions at the edges. We can then find the optimal Q-values by dynamic programming, starting from the terminal states at the leaves:

$$q^*(s_t,a_t) = \begin{cases} 0, & \text{if } s_t \text{ is a terminal state} \\ \mathbb{E}[R(s_t,a_t)] + \gamma \max_{a_{t+1}} \mathbb{E}_{S_{t+1}}[q^*(S_{t+1},a_{t+1})], & \text{otherwise,} \end{cases} \tag{2.4}$$

where $\mathbb{E}[R(s_t,a_t)]$ is the expected reward from the transition (s_t,a_t).

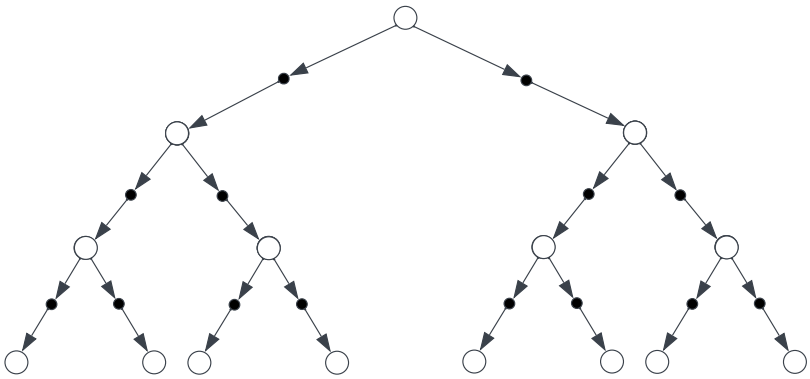


Figure 2.1: A game tree with 2 actions per state and 3 steps per episode. Each open circle represents a state node. Each solid circle represents a state-action pair. The initial state is the root node at the top. The terminal states are at the bottom.

For larger environments, it is intractable to construct the whole game tree. For example, a deterministic environment with K actions per state and T steps per episode would have a game tree with K^T leaves (see Figure 2.1). The Monte-Carlo tree search instead constructs only a part of the game tree.

The MCTS algorithm starts with just one node inside the search tree: the root node contains the state at which we want to find the optimal action. In addition to the state, each node contains the estimated Q-values $\hat{q}(a)$ and visit counts $N(a)$ for visited actions at the node (in these notations, we omit the node index).

The algorithm repeats three subroutines (Figure 2.2):

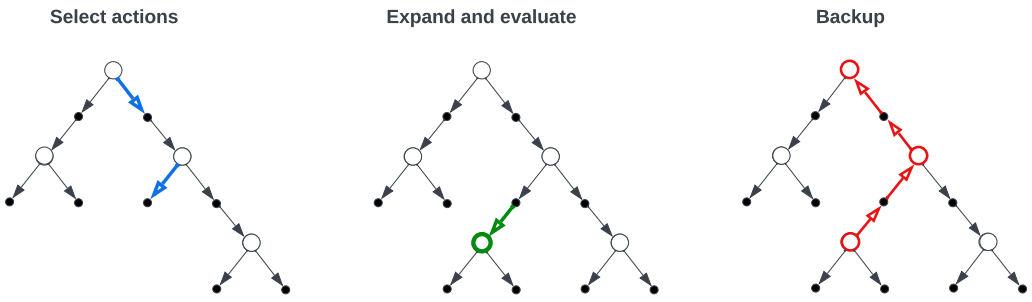


Figure 2.2: The subroutines in Monte-Carlo tree search. The root of the search tree is at the top. The algorithm selects actions in a simulation, expands and evaluates a newly visited state, and backups the new information.

Select actions. To obtain new information, the search has to visit a new state. MCTS does that by selecting actions in a simulation. Many strategies can be used to balance exploration and exploitation. In the next subsections, we will explain the action selection by UCT and AlphaZero.

Expand and evaluate. After visiting a new state, a new leaf node is added to the search tree. The value of the node can be evaluated by a Monte-Carlo return or by a learned value network.

Backup. After adding the new leaf, the tree needs to be updated to help the next action selection. For the visited state-action pairs, the visit counts $N(a)$ are incremented and the estimated Q-values are updated to hold the average return:

$$\hat{q}(a) := \frac{\hat{q}(a)(N(a)-1)+G}{N(a)}, \tag{2.5}$$

where G is an estimate of the return (e.g., the discounted sum of rewards).

Algorithm 1 shows the generic structure of the search. After the search, the FINALACTIONSELECTION selects an action to execute in the environment. The environment then transitions to a new state and a new tree search can start from there.

Algorithm 1 Generic Monte-Carlo tree search

Require: s_0: root state.
Require: n: number of simulations.
  $tree \leftarrow$ INITIALIZETREE(s_0)
  for n simulations **do**
    $node \leftarrow$ GETROOT$(tree)$
    $a \leftarrow$ ACTIONSELECTION$(node)$
    while ISVISITED$(node, a)$ **do**
      $node \leftarrow$ GETCHILD$(node, a)$
      $a \leftarrow$ ACTIONSELECTION$(node)$
    $newLeaf \leftarrow$ EXPAND$(node, a)$
    $tree \leftarrow$ BACKUP$(tree, newLeaf)$
  return FINALACTIONSELECTION(GETROOT$(tree)$)

Each node in the search tree can have a different policy for the action selection. Because these policies are not restricted to share parameters, they can be updated quickly, without degrading the policies in other nodes. We will focus on two pop-

ular MCTS variants: UCT and AlphaZero. Other MCTS variants can be found in Browne et al. (2012).

2.3.2 UCT

Action selection. UCT (Upper Confidence bounds applied to Trees; Kocsis and Szepesvári, 2006) selects an action based on an upper bound of the Q-values. At a node, the action is selected by

$$\arg\max_a \left[\hat{q}(a) + c_p \sqrt{\frac{\ln(\sum_b N(b))}{N(a)}} \right], \tag{2.6}$$

where $\hat{q}(a)$ is the estimated Q-value of the action a, $c_p > 0$ is a constant, and $N(a)$ is the visit count of the action a at the current node. The $\sum_b N(b)$ sums the visit counts of all possible actions. Because an action with a zero visit count gets an infinite $\frac{1}{N(a)}$ ratio, the UCT action selection starts by visiting each action once.

UCT assumes that $\hat{q}(a) \in [0,1]$. This is satisfied on Go and chess by giving reward one for winning and reward zero for losing.

Leaf evaluation. UCT MCTS evaluates a new leaf by a random rollout until the end of the episode. The obtained Monte-Carlo return is then used to update the Q-values. In the simplest case, the rollout policy is uniformly random. MCTS removes the bias from the suboptimal random rollouts only asymptotically, by using an infinite number of simulations and constructing the full tree.

2.3.3 AlphaZero

AlphaZero (Silver et al., 2018) employs a policy network and a value network to enhance the tree search.

Action selection. Given the $\pi(a)$ probabilities produced by the policy network at the current node, AlphaZero selects an action by

$$\arg\max_a \left[q^?(a) + c_1 \pi(a) \frac{\sqrt{1+\sum_b N(b)}}{1+N(a)} \right], \tag{2.7}$$

where $q^?(a)$ is zero for unvisited actions; otherwise it is the estimated Q-value of the

visited action. The $c_1 > 0$ is a factor independent of a. AlphaZero again assumes that $q^?(a) \in [0,1]$. This assumption will not be needed in our later algorithms.

After the search, AlphaZero samples the final action from an (annealed) categorical distribution based on the visit counts of the root actions.

Leaf evaluation. AlphaZero does not use random rollouts. Instead, AlphaZero uses the value network to provide the initial value estimate for a new leaf node.

Like UCT MCTS, given an infinite number of simulations and $\pi(a) > 0$, AlphaZero would find the optimal action. Given a limited number of simulations, AlphaZero does not guarantee to find the optimal action or to find an improved policy. To get a non-asymptotic guarantee, we will later meet a search algorithm designed to produce a policy improvement (Chapter 5).

Training. The policy network and the value network are trained at states from interactions with the environment. The loss for the policy network π is a Kullback–Leibler divergence:

$$\mathrm{KL}(\pi_{\mathrm{MCTS}}, \pi), \tag{2.8}$$

where π_{MCTS} is a target for the policy network. The target is proportional to the visit counts at the root: $\pi_{\mathrm{MCTS}}(a) = \frac{N(a)}{\sum_b N(b)}$. With enough simulations, the π_{MCTS} serves as the proposed improved policy.

A simple loss for the value network $\hat{v}$ is the mean squared error:

$$(G(s) - \hat{v}(s))^2, \tag{2.9}$$

where $G(s)$ is an unbiased estimate of the return after the state s. Alternatively, the value network can be trained with categorical or distributional reinforcement learning losses (Dabney et al., 2018b).

2.3.4 MuZero

MuZero (Schrittwieser et al., 2020) extends AlphaZero to environments without a resettable simulator. Instead of using a simulator, MuZero uses a learned model inside the search.

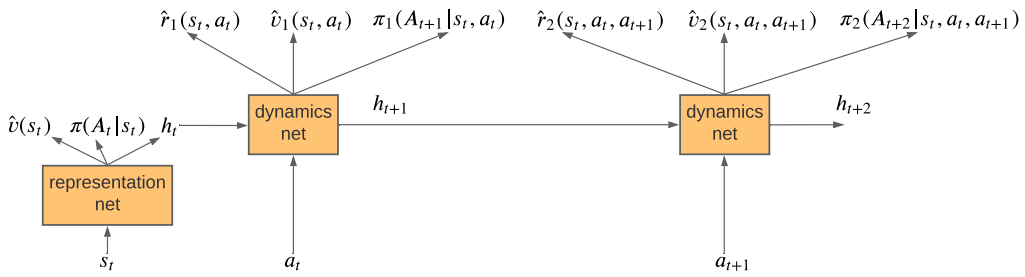


Figure 2.3: The MuZero model. The $\hat{r}_1(s_t,a_t)$ predicts the reward $\mathbb{E}[R_{t+1}|s_t,a_t]$. The $\hat{v}_1(s_t,a_t)$ predicts the expected return $\mathbb{E}[G(S_{t+1})|s_t,a_t]$.

Figure 2.3 shows the MuZero model. The model consists of two networks: a representation network and a dynamics network. The representation network is used at the root of the search tree and provides the value network output $\hat{v}$, the policy network output π, and a state embedding. The embedding vector h_t is a hidden representation of the state s_t.

The networks are not trained to model a state. The networks are trained to model only the quantities needed for planning: rewards, values and policies. The reward prediction $\hat{r}_k(s_t,a_{<t+k})$ models the reward $\mathbb{E}[R_{t+k}|s_t,a_{<t+k}]$. The value prediction $\hat{v}_k(s_t,a_{<t+k})$ models the expected return $\mathbb{E}[G(S_{t+k})|s_t,a_{<t+k}]$. And the policy prediction $\pi_k(A_{t+k}|s_t,a_{<t+k})$ models the policy target $\pi_{\text{MCTS}}(A_{t+k}|s_{t+k})$.

Thanks to the model learning, MuZero is applicable also to Atari and continuous control from pixels (Hubert et al., 2021; Schrittwieser et al., 2021). In the next chapter, we will address MuZero's limitations on stochastic environments.

Chapter 3

Causally Correct Partial Models for Reinforcement Learning

In this chapter, we look for models suitable for planning multiple steps into the future. Such models should give us a distribution or the expected value of the future reward, given a sequence of future actions.

We consider general stochastic environments, and we aim to answer the fundamental questions: *Is it possible to do multi-step planning without generating all pixels? What information should be generated at each step?*

We resolve these questions by introducing a general family of causally correct partial models. These models can be simple and fast, because the modeled sufficient information can be small.

This chapter is relevant only to *multi-step* planning. Shallow, one-step planning does not have problems on stochastic environments, because it does not plan a sequence of actions.

3.1 Introduction

The ability to predict future outcomes of hypothetical decisions is a key aspect of intelligence. A promising approach to capture this ability is via *model-based reinforcement learning* (Munro, 1987; Werbos, 1987; Nguyen and Widrow, 1990; Schmidhuber, 1991). In this framework, the agent learns a model of the environment. One possibility is to learn an action-conditional, next-step model (Oh et al.,

2015; Ha and Schmidhuber, 2018; Chiappa et al., 2017; Xie et al., 2016; Deisenroth and Rasmussen, 2011; Lin and Mitchell, 1992; Li et al., 2015; Diuk et al., 2008; Igl et al., 2018; Ebert et al., 2018; Kaiser et al., 2019; Janner et al., 2019). However, it is often not tractable to accurately model all the available information. This is due both to the fact that conditioning on high-dimensional data such as images would require modeling and generating images in order to plan over several timesteps (Finn and Levine, 2017), and to the fact that modeling images is challenging and may unnecessarily focus on visual details, which are not relevant for acting.

These challenges have motivated researchers to consider simpler models, henceforth referred to as *partial models*, that is, models that are neither conditioned on, nor generate the full set of observed data (Oh et al., 2017; Amos et al., 2018; Guo et al., 2018; Gregor et al., 2019). A notable example of a partial model is the very successful MuZero model (Schrittwieser et al., 2020).

In this chapter, we demonstrate that the commonly used partial models can fail to make correct predictions under a new policy, and we link this failure to a problem in causal reasoning. A key insight of our methodology is the fact that *any piece of information about the state of the environment that is used by the policy to make a decision, but is not available to the model, acts as a confounding variable*. As a result, the learned model is causally incorrect. Using such a model to reason may lead to wrong conclusions about the optimal course of action.

We address these issues of partial models by combining general principles of causal reasoning, probabilistic modeling, and deep learning:

- We identify and clarify a fundamental problem of partial models from a causal-reasoning perspective and illustrate it using simple, intuitive Markov Decision Processes (MDPs) (Section 3.3).

- In order to tackle these shortcomings, we examine the following question: *Is it possible to have a model with a small partial view and still being causally correct in stochastic environments?* We answer affirmatively. We provide the sufficient condition for a causally correct partial model and propose a simple practical way to implement such models (Section 3.5).

- We empirically demonstrate the proposed causal partial models and their benefits in illustrative environments (simple MDPs and stochastic MiniPacman) (Section 3.6).

3.2 A taxonomy of reinforcement learning models

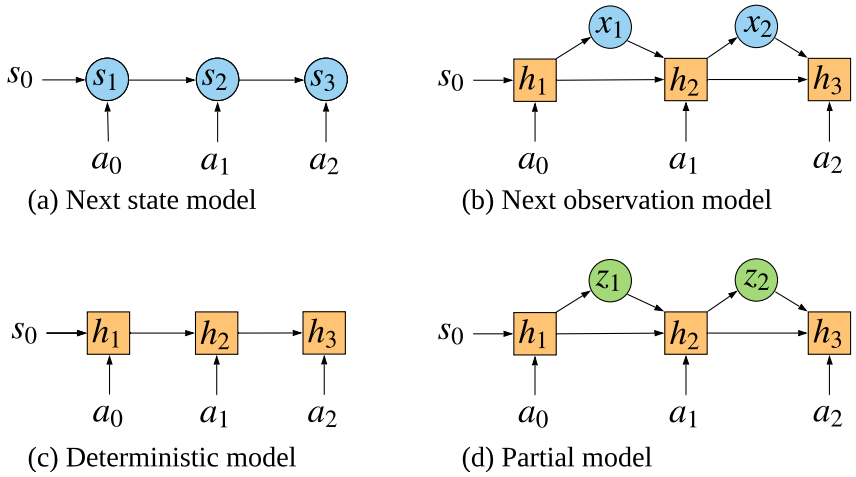
We will classify the commonly used models based on the generated quantities.

Next-step models. A next-step model generates the next state (Figure 3.1a) or the next observation (Figure 3.1b). The generated observation x_t is then fed to the next model state $h_{t+1} = \text{RNN}_h(h_t, x_t, a_t)$ to move a simulation forward. The next observation model learns a distribution $\hat{p}(x_{t+1}|h_t, a_t)$. The model can be also trained to predict other quantities y_t, given the future model state: $\hat{p}(y_t|h_t)$ (e.g., predicting the reward after a sequence of steps).

The conditioning on the generated observations supports learning a model that mimics the environment. However the model can accumulate generated errors if the model is not perfect. In complex environments, the next-step models are rarely perfect. When using images as observations, the model would have to model the joint distribution of the pixels. Such a complex model becomes expensive and slow to run. An example of a next-step model is the action-conditional video prediction by Oh et al. (2015).

Deterministic models. A deterministic model does not generate any stochastic variables during a simulation (Figure 3.1c). The model conditions on the simulation start state s_0 and a sequence of actions. The model can then predict the expected quantities (e.g., reward and value) given the action history. By not generating the observations, the deterministic model can be fast and avoids conditioning on possibly corrupted generated observations. An example of a deterministic model is MuZero (see Figure 2.3).

We consider only planning with primitive actions. We do not consider abstract deterministic models conditioned on no actions (e.g., Predictron by Silver et al. (2017b)) or deterministic models conditioned on policies (e.g., a normal-form representation of a game). These abstract models are less developed.



(a) Next state model (b) Next observation model

(c) Deterministic model (d) Partial model

Figure 3.1: Models used in reinforcement learning. Circles are stochastic nodes; rectangles are deterministic nodes. The model conditions on the actions $a_{<t}$ and models the next state s_t, the next observation x_t, or a partial view z_t.

Partial models. A partial model generates a partial view (Figure 3.1d). The partial view z_t can be any function of the state s_t. E.g., the partial view can be a downsampled observation. In a simulation, the model generates the partial view $z_t \sim \hat{p}(z_t|h_t)$ and moves the simulation forward by updating the model state: $h_{t+1} = \text{RNN}_h(h_t, z_t, a_t)$. When the partial view is small, the partial model can be much faster than a next-step model. In an extreme, the deterministic model is a special case of the partial model, with an empty partial view.

Unsurprisingly, the deterministic models conditioned on primitive actions are not suitable for multi-step planning in stochastic environments. In Section 3.3, we demonstrate the problems of the deterministic models on a simple stochastic MDP. The partial models can also have problems on stochastic environments if the partial view z_t is not informative enough. In Section 3.5, we specify the information requirement for the partial view. We then introduce several ways to implement such sufficient partial views.

After making the partial models suitable for stochastic environments, we can enjoy their fast speed and use them instead of the slow next-step models.

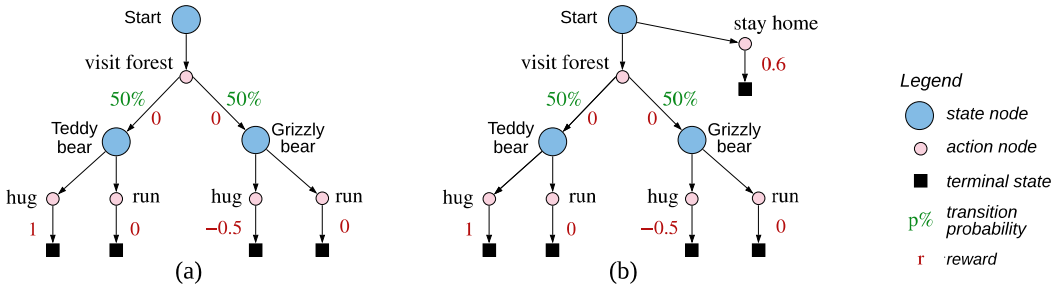


Figure 3.2: Examples of stochastic MDPs. (a) **FuzzyBear:** after visiting a forest, the agent meets either a teddy bear or a grizzly bear with 50% chance and can either hug the bear or run away. (b) **AvoidFuzzyBear:** here, the agent has the extra option to stay home.

3.3 A simple example: FuzzyBear

We will use a simple example to illustrate that the deterministic models are not suitable for multiplestep planning on stochastic environments. Consider the *FuzzyBear* MDP shown in Figure 3.2a: an agent at initial state s_0 transitions into an encounter with either a teddy bear or a grizzly bear with 50% random chance, and can then take an action to either hug the bear or run away. In order to plan, the agent may learn a deterministic model $\hat{p}(r_2|s_0,a_0,a_1)$ that predicts the reward r_2 after performing actions $\{a_0,a_1\}$ starting from state s_0. The deterministic model does not generate and does not condition on the intermediate state s_1, which means the model ignores the observed kind of the bear. The model is suitable for deterministic environments, but it will have problems on stochastic environments, as we shall see. Such a reward model is usually trained on the agent's experience, which consists of sequences of past actions and associated rewards.

Now, suppose the agent wishes to evaluate the sequence of actions $\{a_0 = \textit{visit forest}, a_1 = \textit{hug}\}$ using the average reward under the model $\hat{p}(r_2|s_0,a_0,a_1)$. From Figure 3.2a, we see that the correct average reward is $0.5\times 1+0.5\times(-0.5) = 0.25$. However, if the model has been trained on past experience in which the agent has mostly hugged the teddy bear and run away from the grizzly bear, it will learn that the sequence $\{\textit{visit forest}, \textit{hug}\}$ is associated with a reward close to 1, and that the sequence $\{\textit{visit forest}, \textit{run}\}$ is associated with a reward close to 0. Mathemati-

cally, the model will learn the following conditional probability:

$$p(r_2|s_0,a_0,a_1) = \sum_{s_1} p(s_1|s_0,a_0,a_1)p(r_2|s_1,a_1) \tag{3.1}$$

$$= \sum_{s_1} \frac{p(s_1|s_0,a_0)\pi(a_1|s_1)}{\sum_{s_1'} p(s_1'|s_0,a_0)\pi(a_1|s_1')} p(r_2|s_1,a_1), \tag{3.2}$$

where s_1 is the state corresponding to either *teddy bear* or *grizzly bear*. In the above expression, $p(s_1|s_0,a_0)$ and $p(r_2|s_1,a_1)$ are the transition and reward dynamics of the MDP, and $\pi(a_1|s_1)$ is the agent's behavior policy that generated its past experience. As we can see, the behavior policy affects what the model learns.

The fact that the reward model $\hat{p}(r_2|s_0,a_0,a_1)$ is not robust to changes in the behavior policy has serious implications for planning. For example, suppose that instead of visiting the forest, the agent could have chosen to stay at home, as shown in Figure 3.2b. In this situation, the optimal action is to stay home, because it gives a reward of 0.6, whereas visiting the forest gives at most a reward of $0.5 \times 1 + 0.5 \times 0 = 0.5$. However, an agent that uses the above reward model to plan will overestimate the reward of going into the forest as being close to 1 and will choose the suboptimal action.[1]

One way to avoid this bias is to use a behavior policy that doesn't depend on the state s_1, i.e., $\pi(a_1|s_1) = \pi(a_1)$. Unfortunately, this approach does not scale well to complex environments because it requires an enormous amount of training data for the behavior policy to explore interesting states. A better approach is to make the model robust to changes in the behavior policy. Fundamentally, the problem is due to *causally incorrect reasoning*: the model learns the *observational conditional* $p(r_2|s_0,a_0,a_1)$ instead of the *interventional conditional* given by

$$p(r_2|s_0,\mathrm{do}(a_0),\mathrm{do}(a_1)) = \sum_{s_1} p(s_1|s_0,a_0)p(r_2|s_1,a_1),$$

where the *do-operator* $\mathrm{do}(\cdot)$ means that the actions are performed *independently* of the unspecified context (i.e., independently of s_1). The interventional conditional is

[1] This problem is not restricted to toy examples. In a medical domain, a model could learn that leaving the hospital increases the probability of being healthy.

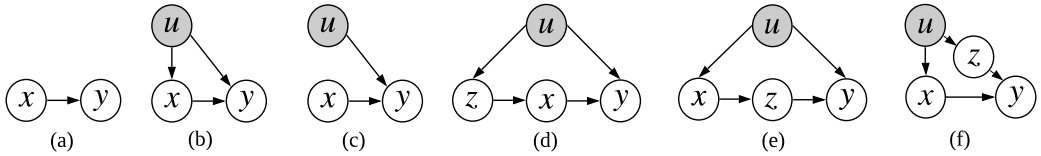


Figure 3.3: Illustration of various causal graphs. (a) Simple dependence without confounding. This is the prevailing assumption in many machine-learning applications. (b) Graph with confounding. (c) Intervention on graph (b) equivalent to setting the value of x and observing y. (d) Graph with a backdoor z blocking all paths from u to x. (e) Graph with a frontdoor z blocking all paths from x to y. (f) Graph with a variable z blocking the direct path from u to y.

robust to changes in the policy and is a more appropriate quantity for planning.

In contrast, the observational conditional quantifies the statistical association between the actions a_0, a_1 and the reward r_2 regardless of whether the actions caused the reward or the reward caused the actions. In Section 3.4, we review relevant concepts from causal reasoning, based on which we propose solutions that address the problem.

Finally, although using $p(r_2|s_0, \text{do}(a_0), \text{do}(a_1))$ leads to causally correct planning, it is not optimal either: it predicts a reward of 0.25 for the sequence $\{\textit{visit forest}, \textit{hug}\}$ and 0 for the sequence $\{\textit{visit forest}, \textit{run}\}$, whereas the optimal policy obtains a reward of 0.5. The optimal policy makes the decision after observing s_1 (teddy bear vs grizzly bear); it is *closed-loop* as opposed to *open-loop*. The solution is to make the intervention at the *policy* level, as we will do in Section 3.5.

3.4 Background on causal reasoning

Causal reasoning provides tools to *answer questions after a change to the data distribution*, without collecting new data.

Many applications of machine learning involve predicting a variable y (target) from a variable x (covariate). A standard way to make such a prediction is by fitting a model $\hat{p}(y|x)$ to a dataset of (x, y) pairs. Then, if we are given a new x and the data-generation process hasn't changed, we can expect that a well trained $\hat{p}(y|x)$ will make an accurate prediction of y.

Confounding: In many situations, however, we would like to use the data to

make different kinds of predictions. For example, what prediction of y should we make if something in the environment has changed, or if we set x ourselves? In these cases, x didn't come from the original data-generation process. This may cause problems in our prediction, because there may be unobserved variables u, known as *confounders*, that affected both x and y during the data-generation process. That is, the actual process was of the form $p(u)p(x|u)p(y|x,u)$ where we only observed x and y as shown in Figure 3.3b.

For example, the x is the action of leaving or staying in a hospital, the y is the obtained happiness, and u can be the unobserved health. With the unobserved u, a model $\hat{p}(y|x)$ fitted on (x,y) pairs will converge to the target $p(y|x) \propto \int p(u)p(x|u)p(y|x,u)du$. However, if at prediction time we set x ourselves (i.e., we tell everyone to leave the hospital now, independently of their health), the actual distribution of the happiness y will be $p(y|\text{do}(x)) = \int p(u)p(y|x,u)du$. This is because setting x ourselves changes the original graph from Figure 3.3b to the one in Figure 3.3c.

Interventions: The operation of setting x to a fixed value $\tilde{x}$ independently of its parents, known as the *do-operator* (Pearl et al., 2016), changes the data-generation process to $p(u)\delta(x-\tilde{x})p(y|x,u)$, where $\delta(x-\tilde{x})$ is the delta-function. As explained above, this results in a different target distribution $\int p(u)p(y|\tilde{x},u)du$, which we refer to as $p(y|\text{do}(x=\tilde{x}))$, or simply $p(y|\text{do}(x))$ when $\tilde{x}$ is implied. Let par_j be the parents of x_j. The do-operator is a particular case of the more general concept of an *intervention*: given a generative process $p(\mathbf{x}) = \prod_j p_j(x_j|\text{par}_j)$, an intervention is defined as a change that replaces one or more factors by new factors. For example, the intervention $p_k(x_k|\text{par}_k) \to \psi_k(x_k|\text{par}'_k)$ replaces $p_k(x_k|\text{par}_k)$ by $\psi_k(x_k|\text{par}'_k)$. This changes the joint distribution $p(\mathbf{x})$ to $p(\mathbf{x})\frac{\psi_k(x_k|\text{par}'_k)}{p_k(x_k|\text{par}_k)}$. The do-operator is a "hard" intervention whereby we replace a factor by a delta function; that is, $p(\mathbf{x}_{/k}, \text{do}(x_k=\tilde{x}_k)) = p(\mathbf{x})\frac{\delta(x_k-\tilde{x}_k)}{p_k(x_k|\text{par}_k)}$, where $\mathbf{x}_{/k}$ denotes the collection of all variables except x_k.

Backdoors and frontdoors: In general, for graphs of the form of Figure 3.3b, $p(y|x)$ does not equal $p(y|\text{do}(x))$. As a consequence, it is not generally possible to

recover $p(y|\text{do}(x))$ using observational data, i.e., (x,y) pairs sampled from $p(x,y)$, regardless of the amount of data available or the expressivity of the model. However, recovering $p(y|\text{do}(x))$ from observational data alone becomes possible if we assume additional structure in the data-generation process. Suppose there exists another observed variable z that blocks all paths from the confounder u to the covariate x, as shown in Figure 3.3d. This variable is a particular case of the concept of a *backdoor* (Pearl et al., 2016, Chapter 3.3) and is said to be a backdoor for the pair $x - y$. In this case, we can express $p(y|\text{do}(x))$ entirely in terms of distributions that can be obtained from the observational data as

$$p(y|\text{do}(x)) = \mathbb{E}_{p(z)}[p(y|z,x)]. \tag{3.3}$$

This formula holds as long as $p(x|z) > 0$ and is referred to as *backdoor adjustment*. The same formula applies when z blocks the effect of the confounder u on y as in Figure 3.3f. More generally, we can use $p(z)$ and $p(y|z,x)$ to compute the marginal distribution $p(y)$ under an arbitrary intervention of the form $p(x|z) \to \psi(x|z)$ on the graph in Figure 3.3d. We refer to the new marginal as $p_{\text{do}(\psi)}(y)$ and obtain it by

$$p_{\text{do}(\psi)}(y) = \mathbb{E}_{p(z)\psi(x|z)}[p(y|z,x)]. \tag{3.4}$$

A similar formula can be derived when there is a variable z blocking the effect of x on y, which is known as a *frontdoor*, shown in Figure 3.3e. Derivations for the backdoor and frontdoor adjustment formulas are provided in Appendix A.1.

Causally correct models: Given data generated by an underlying generative process $p(\mathbf{x})$, we say that a learned model $\hat{p}(\mathbf{x})$ is causally correct with respect to a set of interventions $\mathcal{I}$ if the model remains accurate after any intervention in $\mathcal{I}$. That is, if $\hat{p}(\mathbf{x}) \approx p(\mathbf{x})$ and $\hat{p}(\mathbf{x})$ is causally correct with respect to $\mathcal{I}$, then $\hat{p}_{\text{do}(\psi)}(\mathbf{x}) \approx p_{\text{do}(\psi)}(\mathbf{x})$ for all $\text{do}(\psi)$ in $\mathcal{I}$.

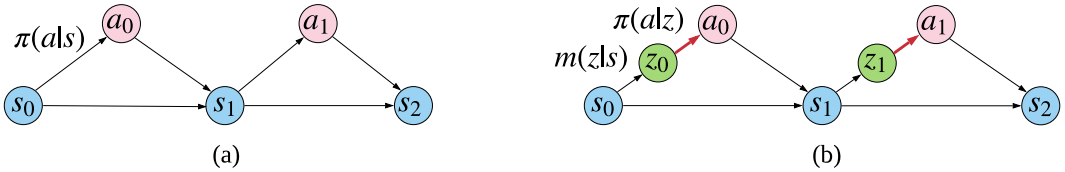


Figure 3.4: The causal diagram for the states and actions in reinforcement learning. (a) An agent interacting with the environment, collecting a trajectory $\{s_t, a_t\}_{t=0}^{T}$. These trajectories are the training data for the models. (b) Same as (a) but also including the backdoor z_t in the collected trajectory. The red arrows indicate the locations of the possible interventions.

3.5 Designing causally correct models

In reinforcement learning (RL), we know the causal diagram for the states and actions (Figure 3.4a). The direction of time defines the causality there.

Equipped with the causality tools, we can find the problem in the design of the deterministic model and propose a fix. We start by looking at the model of some property y_t (e.g., the reward) of the state s_t. The deterministic model learns the conditional distribution $p(y_2|s_0, a_0, a_1)$, given the starting state s_0 and a sequence of actions.[2] By inappropriate design, the deterministic model does not condition on the state s_1. The state s_1 then acts as a confounder (see Figure 3.4a). With the presence of the confounder, we cannot use the model after an intervention on a_1, because $p(y_2|s_0, a_0, \text{do}(a_1))$ is not equal to $p(y_2|s_0, a_0, a_1)$.

Let's design a better model. We want to have a model that can be trained on data collected when following a behavior policy, and we want to be able to reuse the model to answer queries after interventions on some actions. The interventions on the actions can produce the actions from a different distribution than the behavior policy. If we consider a general partial model as in Figure 3.1d, we want to be able to answer a query: $p(y_{t+1}|h_t, z_t, \text{do}(a_t))$. That is, we want to be able to predict the y_{t+1} after an intervention on a_t, given the model state h_t and the partial view z_t. The model state h_t contains a starting state s_0, the previous partial views $z_{<t}$, and the previous actions $a_{<t}$.

To have a model usable after an intervention on an a_t, we will use a carefully

[2] We reindex time for notational simplicity. The state passed to the model is denoted as s_0, even though the state may be in the middle of an episode.

designed partial model. We must be careful to put enough information to the partial views z_t in the collected $\{s_t, z_t, a_t\}_{t=0}^{T}$ trajectories. The collected training data for the model come from a behavior policy, which may be different from a later used simulation policy. We will design a model where $p(y_{t+1}|h_t, z_t, \text{do}(a_t))$ is equal to $p(y_{t+1}|h_t, z_t, a_t)$, and where the model can learn the $p(y_{t+1}|h_t, z_t, a_t)$ distribution from the collected data.

The main theorem: To design a partial model that is causally correct with respect to interventions on the action, it is *sufficient* to use a partial view which makes the behavior action a_t conditionally independent of the state s_t, given the partial view z_t and the model state h_t.

Proof. $a_t \perp s_t | h_t, z_t \implies p(y_{t+1}|h_t, z_t, \text{do}(a_t)) = p(y_{t+1}|h_t, z_t, a_t)$. □

Intuitively, the conditional independence of a_t and s_t makes $p(s_t|h_t, z_t, a_t)$ equal to $p(s_t|h_t, z_t)$ and prevents the conditioning on the action to affect the probability of the previous state. Therefore, causality can be respected. The marginal distribution

$$p(y_{t+1}|h_t, z_t, a_t) = \sum_{s_t} p(s_t|h_t, z_t, a_t) p(y_{t+1}|s_t, a_t) \tag{3.5}$$

is then the same as $p(y_{t+1}|h_t, z_t, \text{do}(a_t)) = \sum_{s_t} p(s_t|h_t, z_t) p(y_{t+1}|s_t, a_t)$.

Deterministic model corollary: In deterministic environments, it is sufficient to use a deterministic model with an empty partial view, because the state s_t can be fully determined from the model state h_t.

Observation model corollary: If the state s_t is represented by the history of actions and observations, it is sufficient to use the observation as the partial view z_t. The model state h_t and z_t will then together form the history of actions and observations.

In the rest of the chapter, we refer to the causally correct partial models as *Causal Partial Models* (CPM).

3.5.1 Sufficient partial views

Not all partial views are sufficient to give us a causally correct partial model. For example, a downsampled observation may be missing an important bullet.

Table 3.1: Comparison between a deterministic model and a causal partial model. The shaded cells indicate the key differences in architectures.

	Deterministic model		**Causal partial model**	
Agent			Partial view $z_t \sim m(z_t \mid s_t)$	
	Action $a_t \sim \pi(a_t \mid s_t)$		Action $a_t \sim \pi(a_t \mid z_t)$	
Model generates	State init. $h_1 = g(s_0, a_0)$		State init. $h_1 = g(s_0, a_0)$	
			Partial view $z_t \sim \hat{p}(z_t \mid h_t)$	
	State update $h_{t+1} = \text{RNN}_h(h_t, a_t)$		State update $h_{t+1} = \text{RNN}_h(h_t, z_t, a_t)$	
	Prediction $y_t \sim \hat{p}(y_t \mid h_t)$		Prediction $y_t \sim \hat{p}(y_t \mid h_t)$	

To have a sufficient partial view, we propose a simple construction. In RL, we usually have access to the internal computation of the behavior policy, so we can choose as the partial view a layer from the behavior policy computation graph. We propose to **use as the partial view a layer that separates the state s_t and the executed action a_t**. The action a_t will be then conditionally independent of the state s_t, given the partial view z_t. Such a partial view acts as a backdoor for the pair $s_t - a_t$.

The location of the proposed partial view is displayed in Figure 3.4b. When collecting data, the agent first produces a partial view $z_t \sim m(z_t \mid s_t)$ and then selects an action $a_t \sim \pi(a_t \mid z_t)$ based only on the partial view. The change made to the agent computation graph is shown in Table 3.1.

We will now list concrete examples of sufficient partial views:

State: We can use the state s_t as the partial view z_t.

Policy probabilities: The z_t can be the probabilities or sufficient statistics produced by the behavior policy. For example, when the actions are discrete, the vector of policy probabilities can be produced by a Dirichlet distribution $m(z_t \mid s_t)$. The $\hat{p}(z_t \mid h_t)$ model can then be a mixture of Dirichlet distributions and can be trained to minimize $\text{KL}(m(z_t \mid s_t) \parallel \hat{p}(z_t \mid h_t))$.

Intended action: The z_t can be the *intended* action before using some form of exploration (e.g., ε-greedy exploration). This is an interesting choice when the actions are discrete, as it is simple to model and, when doing planning, results in a low branching factor that is independent of the complexity of the environment (e.g.,

Algorithm 2 Model training

Data collection on an actor:
For each step:
$z_t \sim m(z_t|s_t)$ ... sample the partial view (e.g., the intended action)
$a_t \sim \pi(a_t|z_t)$ ... sample the executed action (e.g., add ε-exploration)
Collect:
s_t ... agent state
z_t ... partial view
a_t ... executed action
y_{t+1} ... targets (rewards, returns, ...)

Model training on a learner:
Require a trajectory: $s_0, a_{<T}, z_{<T}, y_{\leq T}$
$h_1 = g(s_0, a_0)$ ... initialize the model state
For each trajectory step:
Train $\hat{p}(y_t|h_t)$ to model y_t.
Train $\hat{p}(z_t|h_t)$ to model z_t.
$h_{t+1} = \text{RNN}_h(h_t, z_t, a_t)$ ... update the model state

in visually rich 3D environments).

Combinations: It is possible to combine a layer with additional information. For example, the z_t can contain the intended action, combined with a small down-sampled observation. When a partial view is sufficient, adding more information to it will keep it sufficient.

The model training is summarized in Algorithm 2. When the z_t is the *intended* action before ε-exploration, the z_t will be sampled from a policy $m(z_t|s_t)$ and the *executed* action a_t will then be sampled from an ε-exploration policy $\pi(a_t|z_t) = (1-\varepsilon)\mathbb{I}\{z_t = a_t\} + \varepsilon\frac{1}{n_a}$, where n_a is the number of actions, ε is in $(0,1)$, and the indicator $\mathbb{I}\{z_t = a_t\}$ is 1 if $z_t = a_t$, and zero otherwise. It is imperative that we use some form of exploration to ensure that $\pi(a_t|z_t) > 0$ for all a_t and z_t as this is necessary to allow the model to learn the effects of the actions.

3.5.2 Interventions at the policy level

So far, we have talked about a_t and z_t collected to form the training data for a model. In a simulation, the z_t would be generated from $\hat{p}(z_t|h_t)$ and the simulation action a_t can be freely chosen.

Inside the simulation, we do not have to do open-loop planning; rather we

Algorithm 3 Using the model to generate a simulation under a new policy ψ

Require an agent state: s_0
$a_0 = \psi(a_0|s_0)$ ... choose the first action
$h_1 = g(s_0, a_0)$ ... initialize the model state
For each trajectory step:
  Predict the wanted targets $\hat{p}(y_t|h_t)$ (e.g., rewards, returns, ...).
  $z_t \sim \hat{p}(z_t|h_t)$ ... generate the partial view
  $a_t \sim \psi(a_t|h_t, z_t)$ ... choose the next action
  $h_{t+1} = \text{RNN}_h(h_t, z_t, a_t)$ ... update the model state

can condition on the h_t and z_t. Instead of doing $\text{do}(a_t)$, we can specify a simulation policy: $\psi(a_t|h_t, z_t)$. To make a prediction, we can perform a simulation under the new policy $\psi(a_t|h_t, z_t)$ by directly applying the backdoor-adjustment formula, Equation (3.4), as follows:

$$p_{\text{do}(\psi(a_t|h_t,z_t))}(y_{t+1}|h_t) = \mathbb{E}_{p(z_t|h_t)\psi(a_t|h_t,z_t)}[p(y_{t+1}|h_{t+1})], \tag{3.6}$$

where the components $p(z_t|h_t)$ and $p(y_{t+1}|h_{t+1})$ with $h_{t+1} = \text{RNN}_h(h_t, z_t, a_t)$ can be learned from observational data produced by the agent. The simulation would first generate $z_t \sim \hat{p}(z_t|h_t)$ and then select an action $a_t \sim \psi(a_t|h_t, z_t)$, conditioned on the generated partial view. The generating of a simulation is summarized in Algorithm 3.

3.5.3 Summary before experiments

We have explained how to design causally correct partial models suitable for stochastic environments. Now is a good time to summarize the explanation.

A deterministic model is suitable for planning on deterministic environments. On stochastic environments, planning with MuZero's deterministic model can produce wrong value estimates. Similarly, planning with a partial model can produce wrong value estimates if the partial view is not informative enough. Interestingly, the informative partial view does not have to contain the whole observation. The partial view will prevent confounding if the partial view contains enough information to reproduce the behavior policy.

During planning, we are not restricted to stick with the behavior policy. The

simulation policy can condition on the starting state s_0 and the following sequence of actions $a_{<t}$ and partial views $z_{\leq t}$. In principle, the ability to simulate the behavior policy is all you need for a policy iteration. However, the proposed policy improvement can be larger if planning also tries policies different from the behavior policy.

We provided a spectrum of sufficient partial views. The intended action is one of the smallest sufficient partial views. It can be combined with any other information to form larger sufficient partial views. The best size of the partial view will depend on the environment and on the used planning algorithm. For example, tree search algorithms would prefer a small partial view, because it provides a small branching factor. Table 3.2 summarizes the properties of deterministic models, causal partial models, and next-step models.

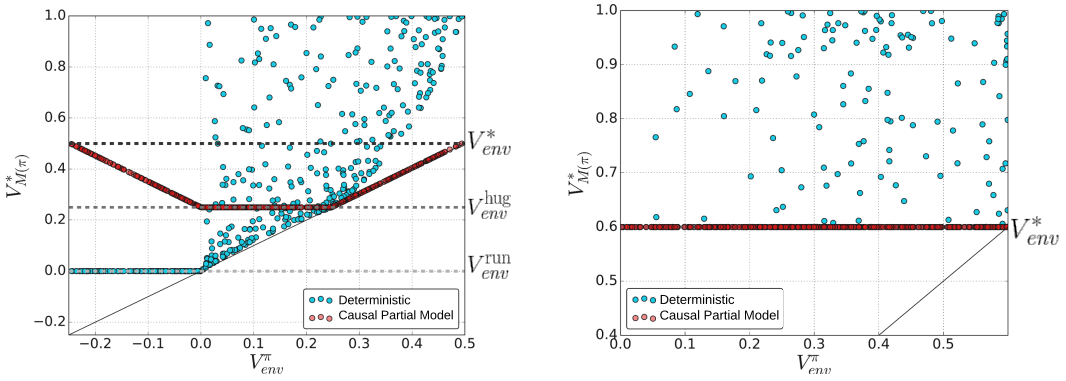
Table 3.2: Models and their properties.

	Deterministic model	**Causal partial model**	**Next-step model**
Generates	nothing	sufficient partial view: z_t	observation
Speed	**fast**	controlled by z_t size	slow
Causally correct	in deterministic environments or with on-policy simulations	**always**	**always**
Simulation variance	**lowest**	controlled by z_t size	high (distracted)
Extra branching	**0**	controlled by z_t size	huge
Invariant of	-	$\pi(a_t\|z_t)$	$\pi(a_t\|s_t)$
Evaluable policies	$\psi(a_t\|s_0, a_{<t})$	$\psi(a_t\|s_0, a_{<t}, z_{\leq t})$	**any**
Training	iterative with policy	iterative with policy	**once**

3.6 Experiments

We will show experiments on simple MDPs and stochastic MiniPacman. For planning, we will use value-iteration, expectimax search, or MuZero's Monte-Carlo tree search.

We focus mainly on making MuZero suitable for stochastic environments; thus the used deterministic model is equivalent to MuZero's model. Additionally, the used causally correct partial model is a minimal modification to MuZero: the partial view is the intended action. The causally correct partial model generates and conditions on the partial view in each simulation.



(a) On FuzzyBear. **(b)** On AvoidFuzzyBear.

Figure 3.5: MDP Analysis: We randomly generate 500 policies and scatter-plot them with x-axis showing the quality of the behavior policy V^π_{env} and y-axis showing corresponding optimal evaluations $V^*_{M(\pi)}$ from dynamic programming inside a model. The unrealistic optimism of the deterministic model evaluations is demonstrated by the blue dots above the maximum possible V^*_{env} line.

3.6.1 Value-iteration analysis on MDPs

In this section, we will analyze the learned models on the FuzzyBear and AvoidFuzzyBear MDPs (Figure 3.2). Because the MDPs are small, we can use a tabular representation for the models and we can compute the exact models. The obtained models are then at the global minimum of their training loss. This will illustrate that the non-causal partial models cannot be fixed by bigger networks or by more training data.

Because the obtained models depend both on the MDP dynamic and on the used behavior policy, we will do the analysis for 500 randomly generated behavior policies. After obtaining a model $M(\pi)$, we can find the optimal simulation policy proposed by planning with the model. For the found optimal simulation policy, we then ask the model to estimate the value $V^*_{M(\pi)}$, of the starting state, under the optimal simulation policy. The exact computation for any episodic MDP is in Appendix A.3. We will see that planning with the deterministic model proposes wrong simulation policies and overestimates their value.

On FuzzyBear MDP (Figure 3.2a), the optimal policy is to always hug the teddy bear and to run away from the grizzly bear. We empirically show the difference between the causal partial models and deterministic models when learning

from randomly generated policies. For each policy, we derive the corresponding converged model $M(\pi)$ equivalent to training on data generated by the policy. We then compute the optimal value of $V^*_{M(\pi)}$ using this model.

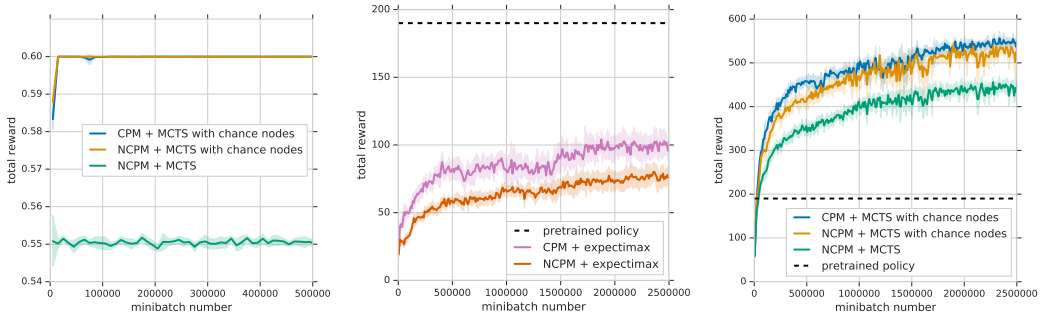
In Figure 3.5a, we see that the causal partial model always produces a value greater than or equal to the value of the behavior policy. The value estimated by the causal partial model can always be achieved in the real environment. If the behavior policy was already good, the simulation policy used inside the model can reproduce the behavior policy by respecting the intended action. If the behavior policy is random, the intended action is uninformative about the underlying state, so the simulation policy has to choose the most rewarding action, independently of the state. Furthermore, if the behavior policy is bad, the simulation policy can choose the opposite of the intended action. This allows the agent to find a very good simulation policy when the behavior policy is very bad. To further improve the policy, the search for better policies should be done also in state s_1. The model can then be retrained on data from the improved policies.

If we look at the deterministic model, we see that it displays the unfortunate property of becoming unrealistically optimistic as the behavior policy becomes better.

On AvoidFuzzyBear MDP (Figure 3.2a), the optimal policy is to stay at home. Indeed, planning with the causal partial model always prefers to stay home, resulting in a constant evaluation for all policies (Figure 3.5b). On the other hand, the deterministic model gives varied, overly optimistic evaluations while choosing the wrong action (visit forest).

3.6.2 Experiments with Expectimax and MCTS

We will now describe experiments with models represented by neural networks and trained by gradient descent. The models will be compared based on their ability to support a tree search. The tree search is used to find an improved policy, given the model. We will use the classical expectimax search (Michie, 1966; Russell and Norvig, 2009) or a variant of MuZero's Monte-Carlo tree search (MCTS) (Schrittwieser et al., 2020). When using a causal partial model, we will use the intended



(a) MCTS on AvoidFuzzyBear. **(b)** Expectimax on MiniPacman. **(c)** MCTS on MiniPacman.

Figure 3.6: (a) MCTS on AvoidFuzzyBear with $p(teddy) = 0.55$. The optimal policy should achieve reward 0.6. (b) The non-causal deterministic model (NCPM) produced visibly worse expectimax search. (c) The causal partial model (CPM) was used inside a MuZero-style MCTS enhanced with chance nodes. The search was able to find a much better policy than the pretrained behavior policy.

action as the partial view z_t. The intended action is a good match for the discrete tree search because the intended action has a small number of categorical values, is easy to model, and has a low variance and a low branching factor.

On AvoidFuzzyBear: On the simple AvoidFuzzyBear MDP (Figure 3.2b), it is enough to use expectimax with a search depth of 3: a decision node, a chance node, and a decision node. The policy found by the search was used to produce the next action for the real environment.

Only the deterministic model was unable to solve the task. Expectimax with the deterministic model consistently preferred the suboptimal stochastic path with the fuzzy bear, as predicted by our theoretical analysis from the previous section. Results with MuZero-style MCTS are in Figure 3.6a. MuZero (NCPM + MCTS) has a number of properties that mitigate the negative effects of the deterministic model. We discuss MuZero properties in Section 3.6.3, and the experimental setup is described in Appendix A.2.

On MiniPacman: We used the 2D MiniPacman environment (Guez et al., 2019) to test whether the non-causal models have problems in other stochastic environments as well. To reduce variance and to remove differences in exploration, we trained all models on data from the same pretrained policy. Indeed, we see in Figures 3.6b and 3.6c that the non-causal deterministic model (NCPM) achieved

visibly smaller reward when used with expectimax or MCTS. Combining MCTS with chance nodes mitigated some of the negative effects of the NCPM, as explained in the next section.

3.6.3 MuZero properties on stochastic environments

MuZero performed surprisingly well on the stochastic environments, even when using the deterministic action-conditioned non-causal model. We will provide an explanation here. First, let us denote by $\hat{v}(s_0, a_0, a_1, \ldots, a_t)$ the output of the learned value network, given $s_0, a_0, a_1, \ldots, a_t$. The $\hat{v}(s_0, a_0)$ will correctly model the expected return, given s_0, a_0. However, the next $\hat{v}(s_0, a_0, a_1)$ can lead to causally incorrect planning because s_1 is a confounder here. When planning with a new policy $\psi(a_1|s_0, a_0)$, the $\sum_{a_1} \psi(a_1|s_0, a_0)\hat{v}(s_0, a_0, a_1)$ can be biased on stochastic environments. The planning with a causally correct model would instead compute the expected return by

$$\sum_{z_1} p(z_1|s_0, a_0) \sum_{a_1} \psi(a_1|s_0, a_0, z_1)\hat{v}(s_0, a_0, z_1, a_1), \tag{3.7}$$

where z_1 is a backdoor to make a_1 independent of s_1, given z_1.

By analyzing the search trees on the AvoidFuzyBear MDP, we were able to find a number of MuZero properties that mitigate the negative effects of the non-causal model:

1. The correctly estimated value $\hat{v}(s_0, a_0)$ discourages opening a tree branch that is suboptimal in the real environment.

2. The learned policy network $\pi(a_t|h_t)$ discourages opening the suboptimal branch if the action leading to the suboptimal branch is not the most probable action. (E.g., MuZero has fewer problems on AvoidFuzzyBear if $p(teddy) < 0.5$.)

3. The averaging of the value-network values from all nodes of the search tree assigns a small weight to the correct $\hat{v}(s_0, a_0)$ only after many simulations.

4. If using chance nodes inside a search tree, the bigger branching factor leads to a shallower search. The shallow search bootstraps more from the causally correct $\hat{v}(s_0, a_0)$.

3.7 Related work

The Book of Why (Pearl and Mackenzie, 2018) discusses in depth the history and motivation behind causality. For a short introduction to causality, see Hardt and Recht (2021). From the vast literature on causality, we will mention only a few related works.

Prior to this work, there has been a growing interest in combining causal inference with reinforcement learning (RL) research in the directions of non-model-based bandit algorithms (Bareinboim et al., 2015; Forney et al., 2017; Zhang and Bareinboim, 2017; Lee and Bareinboim, 2018; Bradtke and Barto, 1996; Lu et al., 2018) and causal discovery with RL (Zhu et al., 2019). Contrary to previous works, in this work we focus on model-based approaches and propose a framework for designing causally correct partial models.

Bottou et al. (2013) provides an excellent example application of the backdoor adjustment to advertisement scoring on Bing. With careful data collection, the effect of an intervention can be estimated by importance sampling on historical data.

Ortega et al. (2021) warns that causal reasoning is important also for the usage of language models. A language model is trained on a data distribution; if the model is then conditioned on text generated from a different distribution, the model would still assume that the text is from the original data distribution. Such careless usage leads to wrong inferences.

In model-based reinforcement learning, MCTS is not the only way to do planning. Our proposed causal partial models can be used equally well inside Dyna (Sutton, 1990), Dyna-2 (Silver et al., 2008), and Policy Gradient Search (Anthony et al., 2019). A sufficient partial view becomes important when combining stochastic environments, multi-step planning, and a simulation policy different from the behavior policy. In Section 3.5.1, we provided examples of sufficient partial views.

In contrast, a lossy encoding of the agent state is not guaranteed to be a sufficient partial view.

Influenced by our work, Stochastic MuZero (Antonoglou et al., 2022) uses VQ-VAE (van den Oord et al., 2017) to produce a discrete partial view. The VQ-VAE encoder is trained end-to-end to produce a partial view that is helpful for predicting the future rewards, values, and policies.

Limited model capacity. While the mentioned causality problems cannot be fixed by larger networks or by longer training, an orthogonal line of work focused on problems caused by limited model capacity. Hallucinated replay (Talvitie, 2014) and Hallucinated DAgger-MC (Talvitie, 2017, 2018) consider imperfect models and provide theoretical guarantees in deterministic environments. To avoid model errors, selective Dyna-style planning (Abbas et al., 2020) truncates the rollout length if the model uncertainty is high. A similar truncation can be used on stochastic environments, to truncate the rollout length if the model does not have enough information to reproduce the behavior policy.

3.8 Conclusion

We showed that the commonly used deterministic or partial models produce wrong conclusions with multi-step planning on stochastic environments. We then explained that partial models can be used for planning if the partial view is informative enough. We defined the needed information and provided a spectrum of sufficient partial views. Hopefully, people will become more careful when applying a model to a changed situation. Our explanation should help them to design causally correct partial models.

The theory is simpler if we do not use multi-step planning. We then do not need to worry about causally correct planning. The one-step Q-values $q(s,a)$ are causally correct with respect to interventions on the action. As we will see in the next chapter, a policy update based on the Q-values is a strong baseline.

Chapter 4

Muesli: Combining Improvements in Policy Optimization

In this chapter, we propose a novel policy update that combines regularized policy optimization with model learning as an auxiliary loss. The update (henceforth Muesli) matches MuZero's state-of-the-art performance on Atari. Notably, Muesli does so without using deep search: it acts directly with a policy network and has computation speed comparable to model-free baselines. The Atari results are complemented by extensive ablations, and by additional results on continuous control and 9x9 Go.

4.1 Introduction

Reinforcement learning (RL) is a general formulation for the problem of sequential decision-making under uncertainty, where a learning system (the *agent*) must learn to maximize the cumulative *rewards* provided by the world it is embedded in (the *environment*), from the experience of interacting with such an environment (Sutton and Barto, 2018). An agent is said to be *value-based* if its behavior (i.e., its *policy*) is inferred (e.g., by inspection) from learned *value* estimates (Sutton, 1988; Watkins, 1989; Rummery and Niranjan, 1994; Tesauro, 1995). In contrast, a *policy-based* agent directly updates a (parametric) policy (Williams, 1992; Sutton et al., 2000) based on past experience. We may also classify as *model-free* the agents that update values and policies directly from experience (Sutton, 1988), and as *model-based*

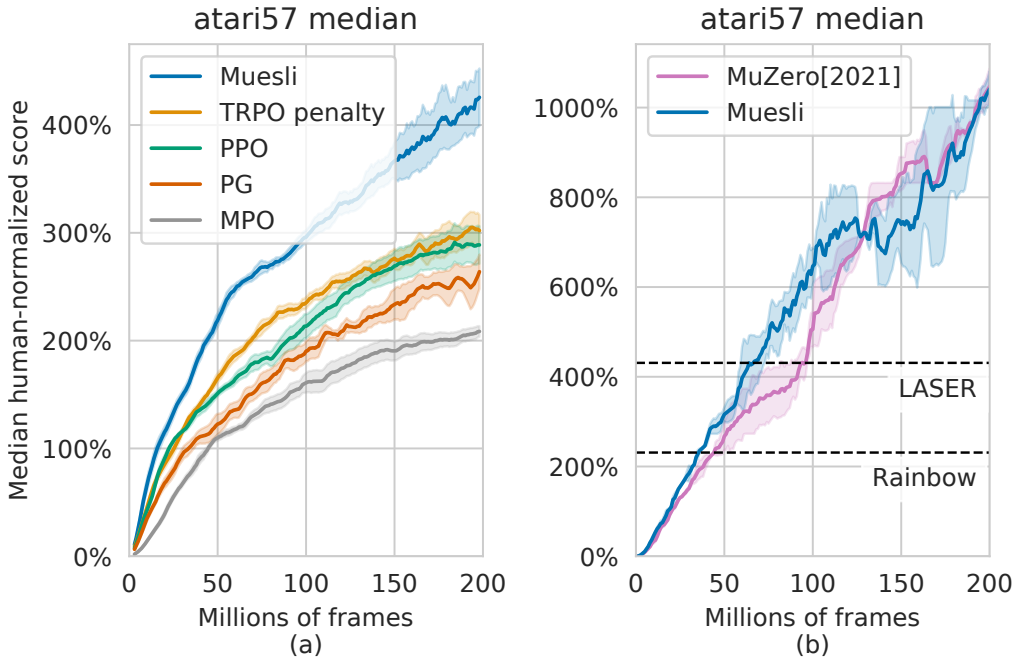


Figure 4.1: Median human-normalized score across 57 Atari games. **(a)** Muesli and other policy updates; all these use the same IMPALA network and a moderate amount of replay data (75%). Shades denote standard errors across 5 seeds. **(b)** Muesli with the larger MuZero network and the high replay fraction used by MuZero (95%), compared to the latest version of MuZero (Schrittwieser et al., 2021). These large-scale runs use 2 seeds. Muesli still acts directly with the policy network and uses one-step look-aheads in updates.

those that use (learned) models (Oh et al., 2015; van Hasselt et al., 2019) to *plan* either global (Sutton, 1990) or local (Richalet et al., 1978; Kaelbling and Lozano-Pérez, 2010; Silver and Veness, 2010) values and policies. Such distinctions are useful for communication, but, to master the singular goal of optimizing rewards in an environment, agents often combine ideas from more than one of these areas (Hessel et al., 2018; Silver et al., 2016; Schrittwieser et al., 2020).

In this chapter, we focus on a critical part of RL, namely *policy optimization*. We leave a precise formulation of the problem for later, but different policy optimization algorithms can be seen as answers to the following crucial question:

Given data about an agent's interactions with the world,
and predictions in the form of value functions or models,
how should we update the agent's policy?

We start from an analysis of the *desiderata* for general policy optimization.

These include support for partial observability and function approximation, the ability to learn stochastic policies, robustness to diverse environments or training regimes (e.g., off-policy data), and being able to represent knowledge as value functions and models. See Section 4.3 for further details on our desiderata for policy optimization.

Then, we propose a policy update combining regularized policy optimization with model-based ideas so as to make progress on the dimensions highlighted in the desiderata. More specifically, we use a model inspired by MuZero (Schrittwieser et al., 2020) to estimate action values via one-step look-ahead. These action values are then plugged into a modified Maximum a Posteriori Policy Optimization (MPO) (Abdolmaleki et al., 2018) mechanism, based on clipped normalized advantages, that is robust to scaling issues without requiring constrained optimization. The overall update, named Muesli, then combines the clipped MPO targets and policy gradients into a *direct* method (Vieillard et al., 2020) for regularized policy optimization.

The majority of our experiments were performed on 57 classic Atari games from the Arcade Learning Environment (Bellemare et al., 2013; Machado et al., 2018), a popular benchmark for deep RL. We found that, on Atari, Muesli can match the state-of-the-art performance of MuZero, without requiring deep search, but instead acting directly with the policy network and using one-step look-aheads in the updates. To help understand the different design choices made in Muesli, our experiments on Atari include multiple ablations of our proposed update. Additionally, to evaluate how well our method generalizes to different domains, we performed experiments on a suite of continuous control environments (based on MuJoCo and sourced from the OpenAI Gym (Brockman et al., 2016)). We also conducted experiments in 9x9 Go in self-play, to evaluate our policy update in a domain traditionally dominated by search methods.

4.2 Background on regularized policy optimization

A regularized policy optimization algorithm solves the following problem:

$$\arg\max_{\pi} \left(\mathbb{E}_{A\sim\pi(\cdot|s)} \left[\hat{q}_{\pi_{\text{prior}}}(s,A) \right] - \Omega(\pi) \right), \tag{4.1}$$

where $\hat{q}_{\pi_{\text{prior}}}(s,a)$ are approximate Q-values of a π_{prior} policy and $\Omega(\pi) \in \mathbb{R}$ is a regularizer. For example, we may use as the regularizer the negative entropy of the policy $\Omega(\pi) = -\lambda \mathrm{H}[\pi]$, weighted by an entropy cost λ (Williams and Peng, 1991)[1]. Alternatively, we may also use $\Omega(\pi) = \lambda\, \mathrm{KL}(\pi_{\text{prior}}, \pi)$, where π_{prior} is the previous policy, as used in TRPO (Schulman et al., 2015).

Following the terminology introduced by Vieillard et al. (2020), we can then solve Eq. 4.1 by either *direct* or *indirect* methods. If $\pi(a|s)$ is differentiable with respect to the policy parameters, a *direct* method applies gradient ascent to

$$J(s,\pi) = \mathbb{E}_{A\sim\pi(\cdot|s)} \left[\hat{q}_{\pi_{\text{prior}}}(s,A) \right] - \Omega(\pi). \tag{4.2}$$

Using the log derivative trick to sample the gradient of the expectation results in the canonical (regularized) policy gradient update (Sutton et al., 2000).

In *indirect* methods, the solution of the optimization problem (4.1) is found exactly, or numerically, for one state and then distilled into a parametric policy (Hinton et al., 2015). For example, Maximum a Posteriori Policy Optimization (MPO) (Abdolmaleki et al., 2018) uses as regularizer $\Omega(\pi) = \lambda\, \mathrm{KL}(\pi, \pi_{\text{prior}})$, for which the exact solution to the regularized problem is

$$\pi_{\text{MPO}}(a|s) = \pi_{\text{prior}}(a|s) \exp\left(\frac{\hat{q}_{\pi_{\text{prior}}}(s,a)}{\lambda} \right) \frac{1}{z(s)}, \tag{4.3}$$

where $z(s) = \mathbb{E}_{A\sim\pi_{\text{prior}}(\cdot|s)} \left[\exp\left(\frac{\hat{q}_{\pi_{\text{prior}}}(s,A)}{\lambda} \right) \right]$ is a normalization factor that ensures that the resulting probabilities form a valid probability distribution (i.e., they sum up to 1).

[1]The Greek letter λ is used to denote a (Lagrange) multiplier. This usage of λ is unrelated to TD-Lambda returns.

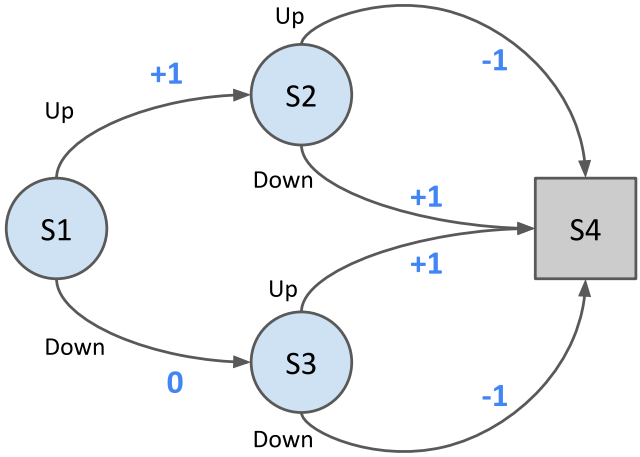


Figure 4.2: An episodic MDP with 4 states. State 1 is the initial state. State 4 is terminal. At each step, the agent can choose amongst two actions: *up* or *down*. The rewards range from -1 to 1, as displayed. The discount is 1. If the state representation $\phi(s)$ is the same in all states, the best stochastic policy is $\pi^*(up|\phi(s)) = \frac{5}{8}$.

4.3 Desiderata and motivating principles

First, to motivate our investigation, we discuss a few desiderata for a general policy optimization algorithm.

4.3.1 Observability and function approximation

Being able to learn stochastic policies, and being able to leverage Monte-Carlo or multi-step bootstrapped return estimates is important for a policy update to be truly general.

This is motivated by the challenges of learning in partially observable environments (Åström, 1965) or, more generally, in settings where function approximation is used (Sutton and Barto, 2018). Note that these two are closely related: if a chosen function approximation ignores a state feature, then the state feature is, for all practical purposes, not observable.

In POMDPs, the optimal memory-less stochastic policy can be better than any memory-less deterministic policy, as shown by Singh et al. (1994). As an illustration, consider the MDP in Figure 4.2; in this problem we have 4 states and, on each step, 2 actions (*up* or *down*). If the state representation of all states is the same $\phi(s) = \varnothing$, the optimal policy is stochastic. We can easily find such a policy with pen and paper: $\pi^*(up|\phi(s)) = \frac{5}{8}$; see Appendix B.2 for details.

It is also known that, in these settings, it is often preferable to leverage Monte-Carlo returns, or at least multi-step bootstrapped estimators, instead of using one-step targets (Jaakkola et al., 1995). Consider again the MDP in Figure 4.2: boostrapping from $v_\pi(\phi(s))$ produces biased estimates of the expected return, because $v_\pi(\phi(s))$ aggregates the values of multiple states; again, see Appendix B.2 for the derivation.

Among the methods in Section 4.2, both policy gradients and MPO allow convergence to stochastic policies, but only policy gradients naturally incorporate multi-step return estimators. In MPO, stochastic return estimates could make the agent overly optimistic ($\mathbb{E}[\exp(G)] \geq \exp(\mathbb{E}[G])$).

4.3.2 Policy representation

Policies may be constructed from action values or they may combine action values and other quantities (e.g., a direct parametrization of the policy or historical data). We argue that the action values alone are not enough.

First, we show that action values are not always enough to represent the best stochastic policy. Consider again the MDP in Figure 4.2 with identical state representation $\phi(s)$ in all states. As discussed, the optimal stochastic policy is $\pi^*(up|\phi(s)) = \frac{5}{8}$. This non-uniform policy cannot be inferred from Q-values, as these are the same for all actions and are thus wholly uninformative about the best probabilities: $q_{\pi^*}(\phi(s), up) = q_{\pi^*}(\phi(s), down) = \frac{1}{4}$. Similarly, a model on its own is also insufficient without a policy, as it would produce the same uninformative action values.

One approach to address this limitation is to parameterize the policy explicitly (e.g., via a policy network). This has the additional advantage that it allows us to directly sample both discrete (Mnih et al., 2016) and continuous (van Hasselt and Wiering, 2007; Degris et al., 2012; Silver et al., 2014) actions. In contrast, maximizing Q-values over continuous action spaces is challenging. Access to a parametric policy network that can be queried directly is also beneficial for agents that act by planning with a learned model (e.g., via MCTS), as it allows to guide search in large or continuous action space.

4.3.3 Robust learning

We seek algorithms that are robust to 1) off-policy or historical data; 2) inaccuracies in values and models; 3) diversity of environments. In the following paragraphs, we discuss what each of these entails.

Reusing data from previous iterations of policy π (Lin, 1992; Riedmiller, 2005; Mnih et al., 2015) can make RL more data efficient. However, if computing the gradient of the objective $\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{q}_{\pi_{\text{prior}}}(s,A)\right]$ on data from an older policy π_{prior}, an unregularized application of the gradient can degrade the value of π. The amount of degradation depends on the *total variation distance* between π and π_{prior}, and we can use a regularizer to control it, as in Conservative Policy Iteration (Kakade and Langford, 2002), Trust Region Policy Optimization (Schulman et al., 2015), and Appendix B.3.

Whether we learn on- or off-policy, agents' predictions incorporate errors. Regularization can also help here. For instance, if Q-values have errors, the MPO regularizer $\Omega(\pi) = \lambda\,\mathrm{KL}(\pi, \pi_{\text{prior}})$ maintains a strong performance bound (Vieillard et al., 2020). The errors from multiple iterations average out, instead of appearing in a discounted sum of the absolute errors. While not all assumptions behind this result apply in an approximate setting, Section 4.5 shows that MPO-like regularizers are helpful empirically.

Finally, robustness to diverse environments is critical to ensure that a policy optimization algorithm operates effectively in novel settings. This can take various forms, but we focus on robustness to diverse reward scales and minimizing problem-dependent hyperparameters. The latter are an especially subtle form of inductive bias that may limit the applicability of a method to established benchmarks (Hessel et al., 2019).

4.3.4 Rich representation of knowledge

Even if the policy is parametrized explicitly, we argue that it is important for the agent to represent knowledge in multiple ways (Degris and Modayil, 2012) to update such a policy in a reliable and robust way. Two classes of predictions have proven particularly useful: value functions and models.

Observability and function approximation	
1a)	Support learning stochastic policies
1b)	Leverage Monte-Carlo targets
Policy representation	
2a)	Support learning the optimal memory-less policy
2b)	Scale to (large) discrete action spaces
2c)	Scale to continuous action spaces
Robust learning	
3a)	Support off-policy and historical data
3b)	Deal gracefully with inaccuracies in the values/model
3c)	Be robust to diverse reward scales
3d)	Avoid problem-dependent hyperparameters
Rich representation of knowledge	
4a)	Estimate values (variance reduction, bootstrapping)
4b)	Learn a model (representation, composability)

Table 4.1: A recap of the *desiderata* or guiding *principles* that we believe are important when designing general policy optimization algorithms. These are discussed in Section 4.3.

Value functions (Sutton, 1988; Sutton et al., 2011) can capture knowledge about a *cumulant* over long horizons, but can be learned with a cost independent of the *span* of the predictions (van Hasselt and Sutton, 2015). They have been used extensively in policy optimization, e.g., to implement forms of variance reduction (Williams, 1992), and to allow updating policies online through bootstrapping, without waiting for episodes to fully resolve (Sutton et al., 2000).

Models can also be useful in various ways: 1) learning a model can act as an auxiliary task (Schmidhuber, 1990; Sutton et al., 2011; Jaderberg et al., 2017; Guez et al., 2020), and help with representation learning; 2) a learned model may be used to update policies and values via planning (Werbos, 1987; Sutton, 1990; Ha and Schmidhuber, 2018); 3) finally, the model may be used to plan for action selection (Richalet et al., 1978; Silver and Veness, 2010). These benefits of learned models are entangled in MuZero. Sometimes it may be useful to decouple them, for instance, to retain the benefits of models for representation learning and policy optimization, without depending on the computationally intensive process of planning for action selection.

4.4 Robust yet simple policy optimization

The full list of *desiderata* is presented in Table 4.1. These are far from solved problems, but they can be helpful to reason about policy updates. In this section, we describe a policy optimization algorithm motivated by these desiderata.

As a basic requirement, the policy update needs to work with a tabular representation. The convergence is harder to guarantee with function approximation. Instead, we practically mitigate divergence by using target networks (Mnih et al., 2015), large networks, and multi-step returns. Large networks are helpful because they have the capacity to give different representations to different states.

4.4.1 Our proposed clipped MPO (CMPO) regularizer

We use the Maximum a Posteriori Policy Optimization (MPO) algorithm (Abdolmaleki et al., 2018) as starting point, since it can learn stochastic policies (1a), supports discrete and continuous action spaces (2c), can learn stably from off-policy data (3a), and has strong performance bounds even when using approximate Q-values (3b). We then improve the degree of control provided by MPO on the total variation distance between π and π_{prior} (3a), avoiding sensitive domain-specific hyperparameters (3d).

MPO uses a regularizer $\Omega(\pi) = \lambda\,\mathrm{KL}(\pi, \pi_{\text{prior}})$, where π_{prior} is the previous policy. Since we are interested in learning from stale data, we allow π_{prior} to correspond to arbitrary previous policies, and we introduce a regularizer $\Omega(\pi) = \lambda\,\mathrm{KL}(\pi_{\text{CMPO}}, \pi)$, based on the new target

$$\pi_{\text{CMPO}}(a|s) = \frac{\pi_{\text{prior}}(a|s)\exp\big(\text{clip}(\hat{\text{adv}}(s,a), -c, c)\big)}{z_{\text{CMPO}}(s)}, \tag{4.4}$$

where $\hat{\text{adv}}(s,a)$ is a non-stochastic approximation of the advantage $\hat{q}_{\pi_{\text{prior}}}(s,a) - \hat{v}_{\pi_{\text{prior}}}(s)$ and the factor $z_{\text{CMPO}}(s)$ ensures the policy is a valid probability distribution. The π_{CMPO} term we use in the regularizer has an interesting relation to natural policy gradients (Kakade, 2001): π_{CMPO} is obtained if the natural gradient is computed with respect to the logits of π_{prior} and then the expected gradient is clipped (for proof, note that the natural policy gradient with respect to the logits is equal to

the advantages (Agarwal et al., 2019)).

The clipping threshold c controls the maximum total variation distance between π_{CMPO} and π_{prior}. Specifically, the total variation distance between π' and π is defined as

$$\mathrm{D}_{\mathrm{TV}}(\pi'(\cdot|s), \pi(\cdot|s)) = \frac{1}{2}\sum_a |\pi'(a|s) - \pi(a|s)|. \tag{4.5}$$

As discussed in Section 4.3.3, constrained total variation supports robust off-policy learning. The clipped advantages allow us to derive not only a bound for the total variation distance but an exact formula:

Theorem 4.4.1 (Maximum CMPO total variation distance)**.** *For any clipping threshold $c > 0$, we have:*

$$\max_{\pi_{\text{prior}}, \hat{\text{adv}}, s} \mathrm{D}_{\mathrm{TV}}(\pi_{\text{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s)) = \tanh(\frac{c}{2}).$$

We refer readers to Appendix B.4 for proof of Theorem 4.4.1; we also verified the theorem predictions numerically.

Note that the maximum total variation distance between π_{CMPO} and π_{prior} does not depend on the number of actions or other environment properties (3d). It depends only on the clipping threshold as visualized in Figure 4.3a. This allows us to control the maximum total variation distance under a CMPO update, for instance, by setting the maximum total variation distance to ε, without requiring the constrained optimization procedure used in the original MPO paper. Instead of the constrained optimization, we just set $c = 2\,\mathrm{arctanh}(\varepsilon)$. We used $c = 1$ in our experiments, across all domains.

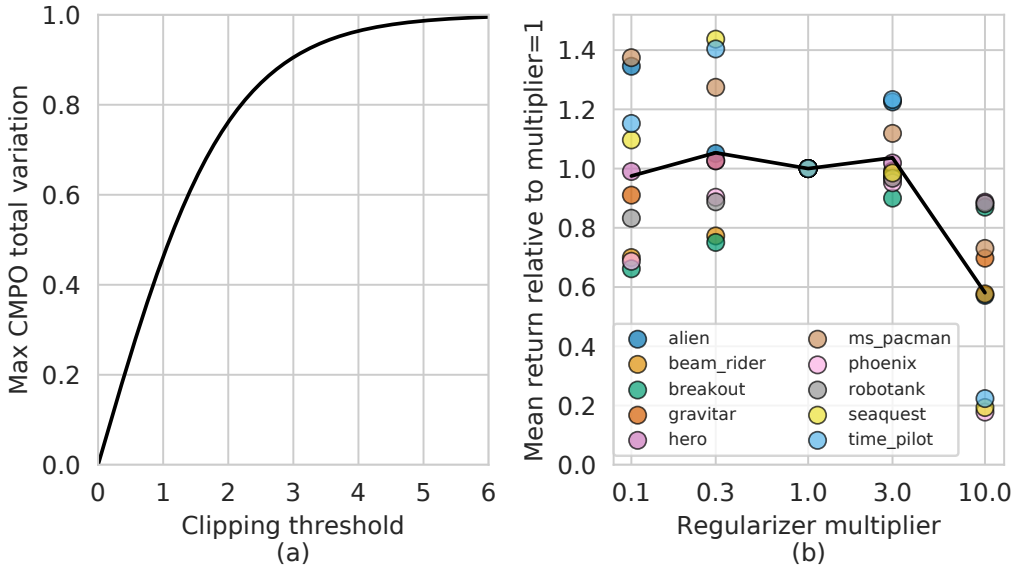


Figure 4.3: **(a)** The maximum total variation distance between π_{CMPO} and π_{prior} is exclusively a function of the clipping threshold c. **(b)** A comparison (on 10 Atari games) of the Muesli sensitivity to the regularizer multiplier λ. Each dot is the mean of 5 runs with different random seeds and the black line is the mean across all 10 games. With Muesli's normalized advantages, the good range of values for λ is fairly large, not strongly problem-dependent, and $\lambda = 1$ performs well on many environments.

4.4.2 A novel policy update

Given the proposed regularizer $\Omega(\pi) = \lambda \, \text{KL}(\pi_{\text{CMPO}}, \pi)$, we can update the policy by *direct* optimization of the regularized objective. That is by gradient descent on

$$L_{\text{PG+CMPO}}(\pi, s) = -\mathbb{E}_{A \sim \pi(\cdot|s)} \left[\hat{\text{adv}}(s, A) \right] + \lambda \, \text{KL}(\pi_{\text{CMPO}}(\cdot|s), \pi(\cdot|s)), \tag{4.6}$$

where the advantage terms in each component of the loss can be normalized using the approach described in Section 4.4.5 to improve the robustness to reward scales.

The first term corresponds to a standard policy gradient update, thus allowing stochastic estimates of $\hat{\text{adv}}(s, A)$ that use Monte-Carlo or multi-step estimators (1b). The second term adds regularization via distillation of the CMPO target, to preserve the desiderata addressed in Section 4.4.1.

Critically, the hyperparameter λ is easy to set (3d), because even if λ is high,

$\lambda\,\mathrm{KL}(\pi_{\mathrm{CMPO}}(\cdot|s),\pi(\cdot|s))$ still proposes *improvements* to the policy π_{prior}. This property is missing in popular regularizers that maximize entropy or minimize a distance from π_{prior}. We refer to the sensitivity analysis depicted in Figure 4.3b for a sample of the wide range of values of λ that we found to perform well on Atari. We used $\lambda = 1$ in all other experiments reported in this chapter.

Both terms can be sampled, allowing us to trade-off the computation cost and the variance of the update; this is especially useful in large or continuous action spaces (2b), (2c).

We can sample the gradient of the first term by computing the loss on data generated on a prior policy π_{prior}, and then use importance sampling to correct for the distribution shift with respect to π. This results in the estimator

$$-\frac{\pi(a|s)}{\pi_b(a|s)}(G^v(s,a)-\hat{v}_{\pi_{\mathrm{prior}}}(s)), \tag{4.7}$$

for the first term of the policy loss. In this expression, $\pi_b(a|s)$ is the behavior policy; the advantage $(G^v(s,a)-\hat{v}_{\pi_{\mathrm{prior}}}(s))$ uses a stochastic multi-step bootstrapped estimator $G^v(s,a)$ and a learned baseline $\hat{v}_{\pi_{\mathrm{prior}}}(s)$.

We can also sample the regularizer by computing a stochastic estimate of the KL on a subset of N actions $a^{(k)}$ sampled from $\pi_{\mathrm{prior}}(s)$. In which case, the second term of Eq. 4.6 becomes (ignoring an additive constant)

$$-\frac{\lambda}{N}\sum_{k=1}^{N}\left[\frac{\exp(\mathrm{clip}(\hat{\mathrm{adv}}(s,a^{(k)}),-c,c))}{z_{\mathrm{CMPO}}(s)}\log\pi(a^{(k)}|s)\right], \tag{4.8}$$

where $\hat{\mathrm{adv}}(s,a)=\hat{q}_{\pi_{\mathrm{prior}}}(s,a)-\hat{v}_{\pi_{\mathrm{prior}}}(s)$ is computed from the learned values $\hat{q}_{\pi_{\mathrm{prior}}}$ and $\hat{v}_{\pi_{\mathrm{prior}}}(s)$. To support sampling just few actions from the current state s, we can estimate $z_{\mathrm{CMPO}}(s)$ for the i-th sample out of N as

$$\tilde{z}^{(i)}_{\mathrm{CMPO}}(s)=\frac{z_{\mathrm{init}}+\sum_{k\neq i}^{N}\exp(\mathrm{clip}(\hat{\mathrm{adv}}(s,a^{(k)}),-c,c))}{N}, \tag{4.9}$$

where z_{init} is an initial estimate. We use $z_{\mathrm{init}}=1$.

4.4.3 Learning a model

As discussed in Section 4.3.4, learning models have several potential benefits. Thus, we propose to train a model alongside policy and value estimates (4b). As in MuZero (Schrittwieser et al., 2020) our model is not trained to reconstruct observations, but is rather only required to provide accurate estimates of rewards, values and policies. It can be seen as an instance of value equivalent models (Grimm et al., 2020).

For training, the model is unrolled $k > 1$ steps, taking as inputs an initial state s_t and an action sequence $a_{<t+k} = a_t, a_{t+1}, ..., a_{t+k-1}$. On each step, the model then predicts rewards $\hat{r}_k$, values $\hat{v}_k$ and policies π_k. Rewards and values are trained to match the observed rewards and values of the states actually visited when executing those actions.

Policy predictions π_k after unrolling the model k steps are trained to match the $\pi_{\text{CMPO}}(\cdot|s_{t+k})$ policy targets computed in the actual observed states s_{t+k}. The policy component of the model loss can then be written as:

$$L_m(\pi, s_t) = \sum_{k=1}^{K} \frac{\text{KL}(\pi_{\text{CMPO}}(\cdot|s_{t+k}), \pi_k(\cdot|s_t, a_{<t+k}))}{K}. \tag{4.10}$$

This differs from MuZero in that here the policy predictions $\pi_k(\cdot|s_t, a_{<t+k})$ are updated towards the targets $\pi_{\text{CMPO}}(\cdot|s_{t+k})$, instead of being updated to match $\pi_{\text{MCTS}}(\cdot|s_{t+k})$ proportional to the visit counts from an MCTS.

4.4.4 Using the model

The first use of a model is as an auxiliary task. We implement this by conditioning the model not on a raw environment state s_t but, instead, on the activations $h(s_t)$ from a hidden layer of the policy network. Gradients from the model loss L_m are then propagated all the way into the shared encoder, to help learning good state representations.

The second use of the model is within the policy update from Eq. 4.6. Specifically, the model is used to estimate the action values $\hat{q}_{\pi_{\text{prior}}}(s, a)$, via *one-step* look-

ahead:

$$\hat{q}_{\pi_{\text{prior}}}(s,a) = \hat{r}_1(s,a) + \gamma \hat{v}_1(s,a), \tag{4.11}$$

and the model-based action values are then used in two ways. First, they are used to estimate the multi-step return $G^v(s,A)$ in Eq. 4.7, by combining action values and observed rewards using the Retrace estimator (Munos et al., 2016). Second, the action values are used in the (non-stochastic) advantage estimate $\hat{\text{adv}}(s,a) = \hat{q}_{\pi_{\text{prior}}}(s,a) - \hat{v}_{\pi_{\text{prior}}}(s)$ required by the regularization term in Eq. 4.8.

Using the model to compute the π_{CMPO} target instead of using it to construct the search-based policy π_{MCTS} has several advantages: a fast analytical formula, stochastic estimation of $\text{KL}(\pi_{\text{CMPO}}(\cdot|s), \pi(\cdot|s))$ in large action spaces (2b), and direct support for continuous actions (2c). In contrast, MuZero's targets π_{MCTS} are only an approximate solution to regularized policy optimization (Grill et al., 2020), and the approximation can be crude when using few simulations.

Note that we could have also used deep search to estimate action-values, and used these in the proposed update. Deep search would however be computationally expensive, and may require more accurate models to be effective (3b).

4.4.5 Normalization

CMPO avoids overly large changes but does not prevent updates from becoming vanishingly small due to *small* advantages. To increase robustness to reward scales (3c), we divide advantages $\hat{\text{adv}}(s,a)$ by the standard deviation of the advantage estimator. A similar normalization was used in PPO (Schulman et al., 2017), but we estimate $\mathbb{E}_{S_t,A_t}\left[(G^v(S_t,A_t) - \hat{v}_{\pi_{\text{prior}}}(S_t))^2\right]$ using moving averages, to support small batches. Normalized advantages do not become small, even when the policy is close to optimal; for convergence, we rely on learning-rate decay.

All policy components can be normalized using this approach, but the model also predicts rewards and values, and the corresponding losses could be sensitive to reward scales. To avoid having to tune, per game, the weighting of these unnormalized components (4c), (4d), we compute losses in a non-linearly transformed space

(Pohlen et al., 2018; van Hasselt et al., 2019), using the categorical reparametrization introduced by MuZero (Schrittwieser et al., 2020).

4.5 An empirical study

In this section, we investigate empirically the policy updates described in Section 4.4. The full agent implementing our recommendations is named Muesli, as an homage to MuZero. The Muesli policy loss is $L_{\text{PG+CMPO}}(\pi, s) + L_m(\pi, s)$. All agents in this section are trained using the Sebulba podracer architecture (Hessel et al., 2021).

First, we use the 57 Atari games in the Arcade Learning Environment (Bellemare et al., 2013) to investigate the key design choices in Muesli by comparing it to suitable baselines and ablations. We use sticky actions to make the environments stochastic (Machado et al., 2018). To ensure comparability, all agents use the same policy network, based on the IMPALA agent (Espeholt et al., 2018). When applicable, the model described in Section 4.4.3 is parametrized by an LSTM (Hochreiter and Schmidhuber, 1997), with a diagram in Figure B.2 in the appendix. Agents are trained using uniform experience replay, and estimate multi-step returns using Retrace (Munos et al., 2016).

In Figure 4.1a, we compare the median human-normalized score on Atari achieved by Muesli to that of several baselines: policy gradients (in red), PPO (in green), MPO (in gray) and a policy gradient variant with TRPO-like $\text{KL}(\pi_b, \pi)$ regularization (in orange). The updates for each baseline are reported in Appendix B.6, and the agents differed only in the policy components of the losses. In all updates we used the same normalization, and trained a MuZero-like model grounded in values and rewards. In MPO and Muesli, the policy loss included the policy model loss from Eq. 4.10. For each update, we separately tuned hyperparameters on 10 of the 57 Atari games. We found the performance on the full benchmark to be substantially higher for Muesli (in blue). In the next experiments we investigate how different design choices contributed to Muesli's performance.

In Figure 4.4, we use the Atari games `beam_rider` and `gravitar` to investi-

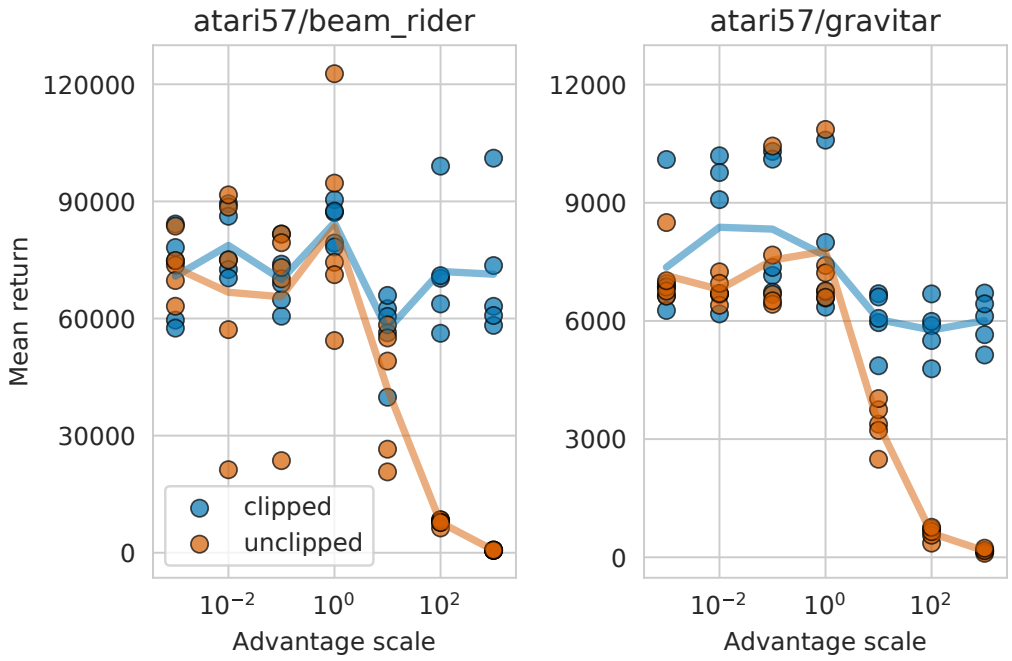


Figure 4.4: A comparison (on two Atari games) of the robustness of clipped and unclipped MPO agents to the scale of the advantages. Without clipping, we found that performance degraded quickly as the scale increased. In contrast, with CMPO, performance was almost unaffected by scales ranging from 10^{-3} to 10^{3}.

gate advantage clipping. Here, we compare the updates that use clipped (in blue) and unclipped (in red) advantages, when first rescaling the advantages by factors ranging from 10^{-3} to 10^{3} to simulate diverse return scales. Without clipping, performance was sensitive to scale, and degraded quickly when scaling advantages by a factor of 100 or more. With clipping, learning was almost unaffected by rescaling, without requiring more complicated solutions such as the constrained optimization introduced in related work to deal with this issue (Abdolmaleki et al., 2018).

In Figure 4.5, we show how Muesli combines the benefits of direct and indirect optimization. A direct MPO update uses the $\lambda\,\mathrm{KL}(\pi, \pi_{\text{prior}})$ regularizer as a penalty; *c.f.* Mirror Descent Policy Optimization (Tomar et al., 2020). Indirect MPO first finds π_{MPO} from Eq. 4.3 and then trains the policy π by the distillation loss $\mathrm{KL}(\pi_{\text{MPO}}, \pi)$. Note the different direction of the KLs. Vieillard et al. (2020) observed that the best choice between direct and indirect MPO is problem-dependent, and we found the same: compare the ordering of direct MPO (in green) and indi-

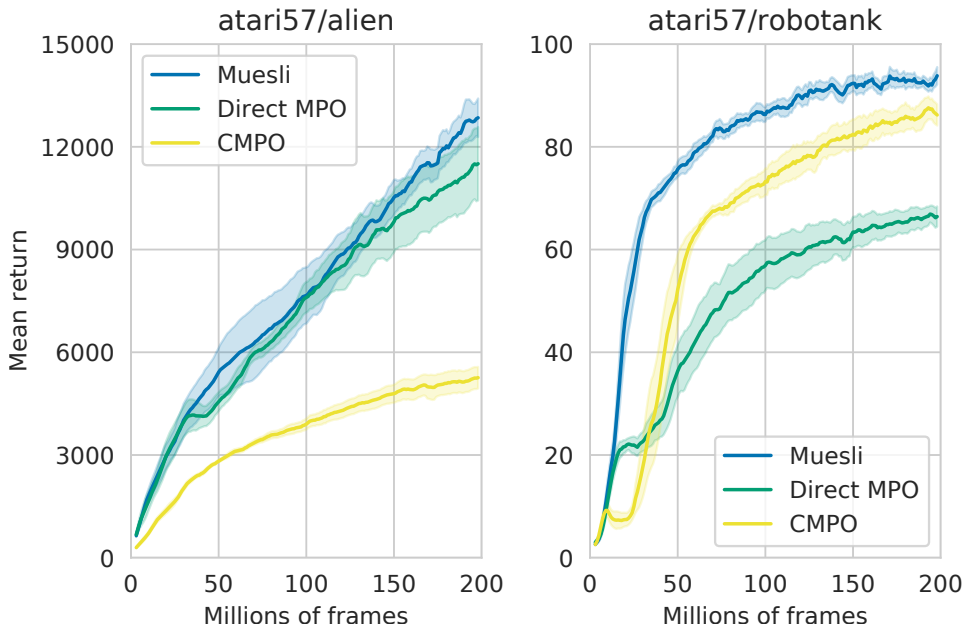


Figure 4.5: A comparison (on two Atari games) of direct and indirect optimization. Whether direct MPO (in green) or indirect CMPO (in yellow) perform best depends on the environment. Muesli, however, typically performs as well as, or better than either one of them. The aggregate score across the 57 games for Muesli, direct MPO and CMPO are reported in Figure B.5 of the appendix.

rect CMPO (in yellow) on the two Atari games `alien` and `robotank`. In contrast, we found that the Muesli policy update (in blue) was typically able to combine the benefits of the two approaches by performing as well as, or better than the best among the two updates on each of the two games. See Figure B.5 in the appendix for aggregate results across more games.

In Figure 4.6a, we evaluate the importance of using multi-step bootstrapped returns and model-based action values in the policy-gradient-like component of Muesli's update. Replacing the multi-step return with an approximate $\hat{q}_{\pi_{\text{prior}}}(s,a)$ (in red in Figure 4.6a) degraded the performance of Muesli (in blue) by a large amount, showing the importance of leveraging multi-step estimators. We also evaluated the role of model-based action value estimates q_π in the Retrace estimator by comparing full Muesli to an ablation (in green) where we instead used model-free values $\hat{v}$ in a V-trace estimator (Espeholt et al., 2018). The ablation performed worse.

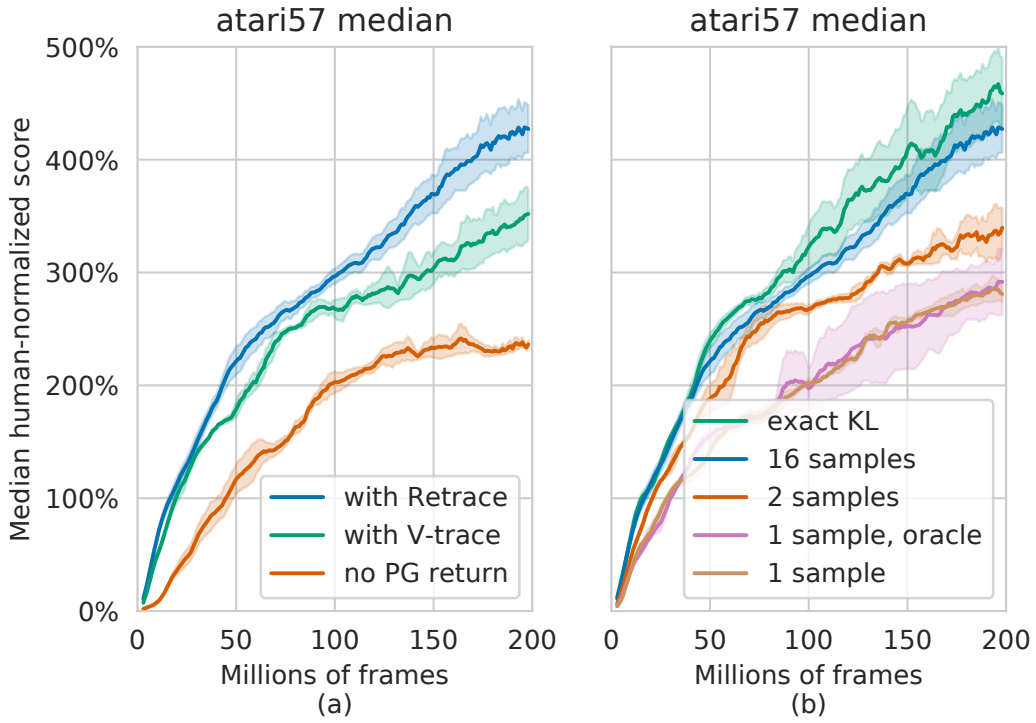


Figure 4.6: Median score across 57 Atari games. **(a)** Return ablations: 1) Retrace or V-trace, 2) training the policy with multi-step returns or with $\hat{q}_{\pi_{\text{prior}}}(s,a)$ only (in red). **(b)** Different numbers of samples to estimate the $\text{KL}(\pi_{\text{CMPO}}, \pi)$. The "1 sample, oracle" (pink) used the exact $z_{\text{CMPO}}(s)$ normalizer, requiring to expand all actions. The ablations were run with 2 random seeds.

In Figure 4.6b, we compare the performance of Muesli when using different numbers of actions to estimate the KL term in Eq. 4.6. We found that the resulting agent performed well in absolute terms ($\sim$ 300% median human-normalized performance) when estimating the KL by sampling as few as a single action (brown). Performance increased by sampling up to 16 actions, which was then comparable to the exact KL.

In Figure 4.7a, we show the impact of different parts of the model loss on representation learning. The performance degraded when training the model to predict only one step (in green). This suggests that training a model to support deeper unrolls (5 steps in Muesli, in blue) is a useful auxiliary task even if using only one-step look-aheads in the policy update. In Figure 4.7a we also show that performance degraded even further if the model was not trained to output policy predictions at each step in the future, as per Eq. 4.10, but instead was trained only to predict rewards

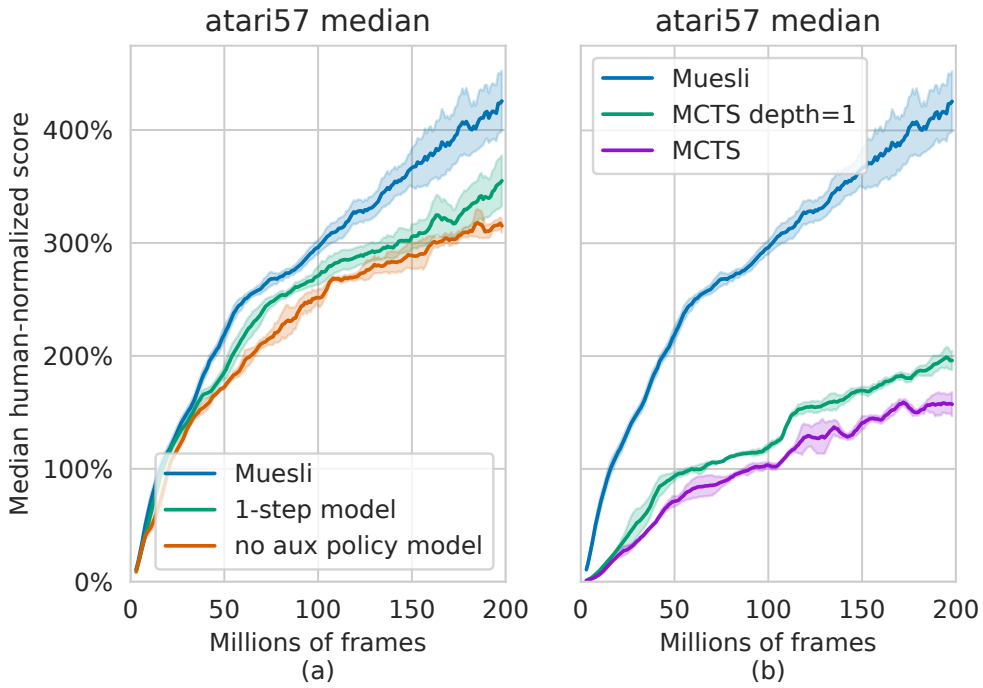


Figure 4.7: Median score across 57 Atari games. **(a)** Muesli ablations that train one-step models (in green), or drop the policy component of the model (in red). **(b)** Muesli and two MCTS-baselines that act by sampling from π_{MCTS} and learn by using π_{MCTS} as the target; all use the IMPALA policy network and an LSTM model.

and values (in red). This is consistent with the value equivalence principle (Grimm et al., 2020): a rich signal from training models to support multiple predictions is critical for this type of model.

In Figure 4.7b, we compare Muesli to an MCTS baseline. As in MuZero, the baseline uses MCTS for both acting and learning. This is not a canonical MuZero, though, as it uses the (smaller) IMPALA network. MCTS (in purple) performed worse than Muesli (in blue) in this regime. We ran another MCTS variant with limited search depth (in green); this performed better than full MCTS, suggesting that with insufficiently large networks, the model may not be sufficiently accurate to support deep search. In contrast, Muesli performed well even with these smaller models (3b).

Since we know from the literature that MCTS can be very effective in combination with larger models, in Figure 4.1b we reran Muesli with a much larger policy network and model, similar to that of MuZero. In this setting, Muesli matched the

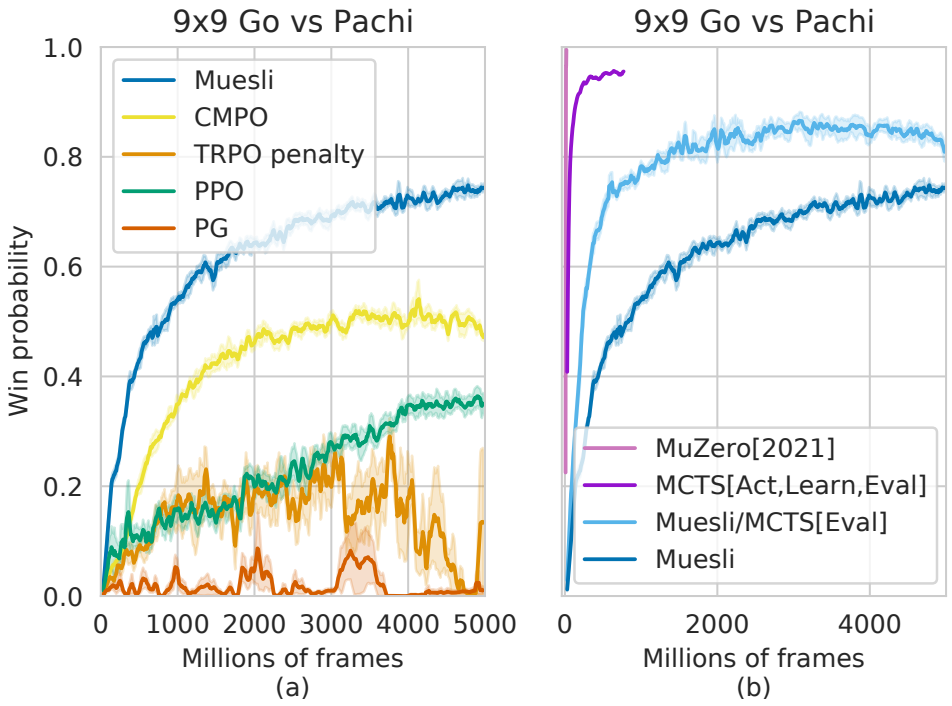


Figure 4.8: Win probability on 9x9 Go when training from scratch, by self-play, for 5B frames. Evaluating 3 seeds against Pachi with 10K simulations per move. **(a)** Muesli and other search-free baselines. **(b)** MuZero MCTS with 150 simulations, and Muesli with and without the use of MCTS at the evaluation time only.

published performance of MuZero (the current state of the art on Atari in the 200M frames regime). Notably, Muesli achieved this without relying on deep search: it still sampled actions from the fast policy network and used one-step look-aheads in the policy update. We note that the resulting median score matches MuZero and is substantially higher than all other published agents, see Table 4.2 to compare the final performance of Muesli to other baselines.

Next, we evaluated Muesli on learning 9x9 Go from self-play. This requires to handle non-stationarity and a combinatorial space. It is also a domain where deep search (e.g., MCTS) has historically been critical to reach non-trivial performance. In Figure 4.8a, we show that Muesli (in blue) still outperformed the strongest baselines from Figure 4.1a, as well as CMPO on its own (in yellow). All policies were evaluated against Pachi (Baudiš and Gailly, 2011). Muesli reached a $\sim$75% win rate against Pachi: to the best of our knowledge, this is the first system to do so from self-play alone without deep search. In the appendix we report even stronger

Table 4.2: Median human-normalized score across 57 Atari games from the ALE, at 200M frames, for several published baselines. These results are sourced from different papers, thus the agents differ along multiple dimensions (e.g., network architecture and amount of experience replay). MuZero and Muesli both use a very similar network, the same proportion of replay, and both use the harder version of the ALE with sticky actions (Machado et al., 2018). The ± denotes the standard error over 2 random seeds.

AGENT	MEDIAN
DQN (Mnih et al., 2015)	79%
DreamerV2 (Hafner et al., 2020)	164%
IMPALA (Espeholt et al., 2018)	192%
Rainbow (Hessel et al., 2018)	231%
Meta-gradient$\{\gamma, \lambda\}$ (Xu et al., 2018)	287%
STAC (Zahavy et al., 2020)	364%
LASER (Schmitt et al., 2020)	431%
MuZero Reanalyse (Schrittwieser et al., 2021)	**1,047** ±40%
Muesli	**1,041** ±40%

win rates against GnuGo (Bump et al., 2005).

In Figure 4.8b, we compare Muesli to MCTS on Go; here, Muesli's performance (in blue) fell short of that of the MCTS baseline (in purple), suggesting there is still value in using deep search for acting in some domains. This is demonstrated also by another Muesli variant that uses deep search at evaluation only. Such a Muesli/MCTS[Eval] hybrid (in light blue) recovered part of the gap with the MCTS baseline, without slowing down training. For reference, with the pink vertical line we depict the published MuZero, with its even greater data efficiency thanks to more simulations, a different network, more replay, and early resignation.

Finally, we tested the same agents on MuJoCo environments in OpenAI Gym (Brockman et al., 2016), to test if Muesli can be effective on continuous domains and on smaller data budgets (2M frames). Muesli performed competitively. We refer readers to Figure B.4, in the appendix, for the results.

4.6 Conclusion

Starting from our desiderata for general policy optimization, we proposed an update (Muesli) that combines policy gradients with Maximum a Posteriori Policy

Optimization (MPO) and model-based action values. We empirically evaluated the contributions of each design choice in Muesli, and compared the proposed update to related ideas from the literature. Muesli demonstrated state-of-the-art performance on Atari (matching MuZero's most recent results), without the need for deep search. Muesli even outperformed MCTS-based agents, when evaluated in a regime of smaller networks and/or reduced computational budgets. Finally, we found that Muesli could be applied out of the box to self-play 9x9 Go and continuous control problems, showing the generality of the update (although further research is needed to really push the state of the art in these domains). We hope that our findings will motivate further research in the rich space of algorithms at the intersection of policy gradient methods, regularized policy optimization and planning.

In the next chapter, we will show how to reach the state of the art even on Go, by allowing a tree search to represent the optimal stochastic policy (desiderata (1a) and (2a)).

Chapter 5

Policy Improvement by Planning with Gumbel

AlphaZero is a powerful reinforcement learning algorithm based on approximate policy iteration and tree search. However, AlphaZero can fail to improve its policy network, if not visiting all actions at the root of a search tree. To address this issue, we propose a policy improvement algorithm based on sampling actions without replacement. Furthermore, we use the idea of policy improvement to replace the more heuristic mechanisms by which AlphaZero selects and uses actions, both at root nodes and at non-root nodes. Our new algorithms, Gumbel AlphaZero and Gumbel MuZero, respectively without and with model-learning, match the state of the art on Go, chess, and Atari, and significantly improve prior performance when planning with few simulations.

5.1 Introduction

In 2018, AlphaZero (Silver et al., 2018) demonstrated a single algorithm achieving state-of-the-art results on Go, chess, and Shogi. The community reacted quickly. Leela Chess Zero (Linscott et al., 2018) was created to reproduce AlphaZero results on chess, winning Top Chess Engine Championship in 2019. Soon, all top-rated classical chess engines replaced traditional evaluation functions with Efficiently Updatable Neural Network (Nasu, 2018).

AlphaZero was itself generalized by MuZero (Schrittwieser et al., 2020).

While AlphaZero requires a black-box model of the environment, MuZero learns an abstract model of the environment. Essentially, MuZero learns the rules of Go, chess, and Shogi from interactions with the environment. This allows MuZero to excel also at Atari and continuous control from pixels (Hubert et al., 2021).

In this work, we redesign and improve AlphaZero. In particular, we consider the mechanisms by which AlphaZero selects and uses actions, which are based upon a variety of heuristic ideas that have proven especially effective in Go, chess, and Atari (Silver et al., 2018; Schrittwieser et al., 2020). However, when using a small number of simulations, some of AlphaZero's mechanisms perform poorly. We use the principle of policy improvement to suggest new mechanisms with a better theoretical foundation. More specifically, we consider each mechanism in turn, alongside our proposed modifications:

- **Selecting actions to search at the root node.** To explore different actions during training, AlphaZero selects actions by adding Dirichlet noise to its policy network, and then performs a search using the perturbed policy. However, this does not ensure a policy improvement. We instead propose to sample actions without replacement by using the Gumbel-Top-k trick (Section 2), and perform a search using the same Gumbel values to influence the selection of the best action (Section 3.3), and show that this guarantees a policy improvement when action-values are correctly evaluated.

- **Selecting actions at the root node.** AlphaZero uses a variant of the PUCB algorithm (Rosin, 2011) to select actions at the root node. This algorithm was designed to optimize cumulative regret in a *bandit-with-predictor* setting (i.e., given prior recommendations from the policy network). However, no ancestors are dependent upon the evaluation of the root node, and the performance of the Monte-Carlo tree search therefore only depends upon the final recommended action at the root node, and not upon the intermediate actions selected during search (Bubeck et al., 2011). Consequently, we propose to use the Sequential Halving algorithm (Karnin et al., 2013) at the root node to optimize simple regret in a stochastic bandit with a predictor (Section 5.3.4).

- **Selecting actions in the environment.** Once search is complete, AlphaZero selects an action by sampling from an (annealed) categorical distribution based upon the visit counts of root actions resulting from the search procedure. We instead propose to select the singular action resulting from the Sequential Halving search procedure.

- **Policy network update.** AlphaZero updates its policy network towards a categorical distribution based upon the visit counts of root actions. However, even if the considered actions are correctly evaluated, this does not guarantee a policy improvement, especially when using small numbers of simulations (Grill et al., 2020). We instead propose a policy improvement based upon the root action values computed during search, and we update the policy network towards that policy improvement (Section 5.4).

- **Selecting actions at non-root nodes.** AlphaZero uses the PUCT algorithm to select actions at non-root nodes. We instead propose to select actions according to a policy improvement (similar to the proposal of Grill et al. (2020)) based upon a completion of the action values. Furthermore, rather than sampling directly from this policy improvement, we propose a deterministic action selection procedure that matches the empirical visit counts to the desired policy improvement (Section 5.5).

The proposed modifications are also applicable to MuZero or any agent with a policy network and an expensive Q-network. The modifications are most helpful when using a small number of simulations, relative to the number of actions. When using a large number of simulations, AlphaZero works well. We tried to ensure that the new search is principled, better with a smaller number of simulations, and never worse. We succeeded on all tested domains: Go, chess, and Atari.

5.2 Background on Gumbel-Max

Before explaining the improved search, we will explain the Gumbel-Max trick and the Gumbel-Top-k trick. The Gumbel-Max trick was popularized by Gumbel-

Softmax for a gradient approximation. In this work, we are not interested in approximate gradients. Instead, we use the Gumbel-Top-k trick to sample without replacement.

Gumbel-Max trick. (Gumbel, 1954; Luce, 1959; Maddison et al., 2017; Jang et al., 2017) Let π be a categorical distribution with logits $\in \mathbb{R}^k$, such that $\text{logits}(a)$ is the logit of the action a. We can obtain a sample A from the distribution π by first generating a vector of k Gumbel variables and then taking argmax:

$$(g \in \mathbb{R}^k) \sim \text{Gumbel}(0) \tag{5.1}$$

$$A = \arg\max_a (g(a) + \text{logits}(a)). \tag{5.2}$$

Gumbel-Top-k trick. (Yellott, 1977; Vieira, 2014; Kool et al., 2019) The Gumbel-Max trick can be generalized to sampling n actions *without replacement*, by taking n top actions:

$$(g \in \mathbb{R}^k) \sim \text{Gumbel}(0) \tag{5.3}$$

$$A_1 = \arg\max_a (g(a) + \text{logits}(a)) \tag{5.4}$$

$$\vdots$$

$$A_n = \arg\max_{a \notin \{A_1, \ldots, A_{n-1}\}} (g(a) + \text{logits}(a)). \tag{5.5}$$

We will denote the set of n top actions by $\texttt{argtop}(g + \text{logits}, n) = \{A_1, A_2, \ldots, A_n\}$.

5.3 Planning at the root

We are interested in improving AlphaZero Monte-Carlo tree search (MCTS). In this section, we will focus on the action selection at the root of the search tree.

5.3.1 Problem setting

Both AlphaZero and MuZero have access to a policy network. At the root of the search tree, they can explore n simulations, before selecting an action for the real environment. We will formalize the problem as a deterministic bandit with a predictor and we will later extend it to a stochastic bandit and MCTS.

Bandit. A k-armed deterministic bandit is a vector of Q-values $q \in \mathbb{R}^k$, such that $q(a)$ is the Q-value of the action a. The agent interacts with the bandit in n simulations (aka rounds). In each simulation $t \in \{1,\dots,n\}$, the agent selects an action $A_t \in \{0,\dots,k-1\}$ and visits the action to observe the Q-value $q(A_t)$.

The objective is to maximize the Q-value from a special last action A_{n+1}. That means we want to maximize $\mathbb{E}[q(A_{n+1})]$. This objective is equivalent to minimization of *simple regret*. The simple regret differs from the *cumulative regret* from all n simulations. Bubeck et al. (2011), Hay and Russell (2011), and Tolpin and Shimony (2012) already argued that at the root of the search tree we care about the simple regret.

The problem becomes interesting when the number of possible actions is larger than the number of simulations, i.e., when $k > n$. For example, 19x19 Go has 362 possible actions, and we will do experiments with as few as $n = 2$ simulations. Fortunately, the policy network can help.

Predictor. In the *bandit-with-predictor* setting (Rosin, 2011), the agent is equipped with a predictor: the policy network. Before any interaction with the bandit, the policy network predicts the best action by producing a probability distribution π. The agent can use the policy network predictions to make more informed decisions.

Policy improvement. Naturally, we would like to have an agent that acts better than, or as well as, the policy network. That is, we would like to obtain a policy improvement. If the agent's action selection produces a *policy improvement*, then

$$\mathbb{E}\left[q(A_{n+1})\right] \geq \sum_a \pi(a) q(a), \tag{5.6}$$

where the probability $\pi(a)$ is the policy network prediction for the action a.[1] The policy network can then keep improving by modeling an improved policy.

[1] Inequality 5.6 can be strict if we assume that an action has a positive advantage and its $\pi(a) > 0$.

5.3.2 Motivating counterexample

We will show that the commonly used heuristics fail to produce a policy improvement.

Example 1. Acting with the best action from the top-n most-probable actions fails to produce a policy improvement. Let's demonstrate that. Let $q = (0,0,1)$ be the Q-values and let $\pi = (0.5, 0.3, 0.2)$ be the probabilities produced by the policy network. The value of the policy network is $\sum_a \pi(a)q(a) = 0.2$. For $n = 2$ simulations, the set of the most-probable actions is $\{0,1\}$. With that, the heuristic would select $A_{n+1} = \arg\max_{a\in\{0,1\}} q(a)$. The expected value of such an action is $\mathbb{E}\left[q(A_{n+1})\right] = 0$, which is worse than the value of the policy network.

You can find other counterexamples by generating random q and π vectors and testing the policy improvement (Inequality 5.6). If the number of simulations is smaller than the number of actions, then UCT and AlphaZero do not ensure a policy improvement.

5.3.3 Planning with Gumbel

We will design a policy improvement algorithm for the deterministic bandit with a predictor π. After n simulations, the algorithm should propose an action A_{n+1} with $\mathbb{E}\left[q(A_{n+1})\right] \geq \sum_a \pi(a)q(a)$.

One possibility is to sample n actions from π, and then to select from the sampled actions the action with the highest $q(a)$. Instead of sampling with replacement, we can reduce the variance by sampling without replacement.

Still, the sampled actions contain a limited amount of information about π. We should exploit the knowledge of π and its logits when selecting A_{n+1}. The main idea is to sample n actions without replacement by using the Gumbel-Top-k trick, and then to use the *same* Gumbel g to select the action with the highest $g(a) + \text{logits}(a) + \sigma(q(a))$. The σ can be any monotonically increasing transformation. The pseudocode for the algorithm is in Algorithm 4.

Algorithm 4 Policy Improvement by Planning with Gumbel

Require: k: number of actions.
Require: $n \leq k$: number of simulations.
Require: logits $\in \mathbb{R}^k$: predictor logits from a policy network π.
  Sample k Gumbel variables:
    $(g \in \mathbb{R}^k) \sim \text{Gumbel}(0)$
  Find n actions with the highest $g(a) + \text{logits}(a)$:
    $\mathcal{A}_{\text{topn}} = \texttt{argtop}(g + \text{logits}, n)$
  Get $q(a)$ for each $a \in \mathcal{A}_{\text{topn}}$ by visiting the actions.
  From $\mathcal{A}_{\text{topn}}$, find the action with the highest $g(a) + \text{logits}(a) + \sigma(q(a))$:
    $A_{n+1} = \arg\max_{a \in \mathcal{A}_{\text{topn}}}(g(a) + \text{logits}(a) + \sigma(q(a)))$
  return A_{n+1}

The algorithm produces a policy improvement because

$$q(\arg\max_{a \in \mathcal{A}_{\text{topn}}}(g(a) + \text{logits}(a) + \sigma(q(a)))) \geq q(\arg\max_{a \in \mathcal{A}\text{topn}}(g(a) + \text{logits}(a))). \quad (5.7)$$

This holds for any Gumbel g, so it holds also for expectations: $\mathbb{E}[q(A_{n+1})] \geq \mathbb{E}_{A \sim \pi}[q(A)]$. The $\arg\max_{a \in \mathcal{A}\text{topn}}(g(a) + \text{logits}(a))$ is equivalent to sampling from the policy network π (see the Gumbel-Max trick or Appendix C.1). By using the same Gumbel vector g in the `argtop` and $\arg\max$, we avoid a double-counting bias.

The prior knowledge contained in the logits can help on partially observable environments, or when working with approximate or stochastic Q-values.

5.3.4 Planning on a stochastic bandit

We can now extend the algorithm to a stochastic bandit. A stochastic bandit provides only a stochastic estimate of the expected Q-value $q(a)$. In that setting, we will use the empirical mean $\hat{q}(a)$ instead of $q(a)$. Obviously, the empirical mean would be better estimated if visiting an action multiple times. We have to choose which actions to visit and how many times. We can control this in two places. First, we can control the number of actions sampled without replacement. Second, we can use a bandit algorithm to efficiently explore the set of sampled actions.

There are multiple bandit algorithms for simple regret minimization. In our preliminary experiments, Sequential Halving (Karnin et al., 2013) was easier to tune than UCB-E (Audibert et al., 2010) and UCB$\sqrt{\cdot}$ (Tolpin and Shimony, 2012).

Conveniently, Sequential Halving does not have problem-dependent hyperparameters.

We present Sequential Halving with Gumbel in Algorithm 5, with an illustration in Figure 5.1. Sequential Halving is used to identify the action with the highest $g(a) + \text{logits}(a) + \sigma(\hat{q}(a))$.

Algorithm 5 Sequential Halving with Gumbel

Require: k: number of actions.
Require: $m \leq k$: number of actions sampled without replacement.
Require: n: number of simulations.
Require: logits $\in \mathbb{R}^k$: predictor logits from a policy network π.
Sample k Gumbel variables:
$(g \in \mathbb{R}^k) \sim \text{Gumbel}(0)$
Find m actions with the highest $g(a) + \text{logits}(a)$:
$\mathcal{A}_{\text{topm}} = \texttt{argtop}(g + \text{logits}, m)$
Use Sequential Halving with n simulations to identify the best action from the $\mathcal{A}_{\text{topm}}$ actions, by comparing $g(a) + \text{logits}(a) + \sigma(\hat{q}(a))$.
$A_{n+1} = \arg\max_{a \in \mathit{Remaining}}(g(a) + \text{logits}(a) + \sigma(\hat{q}(a)))$
return A_{n+1}

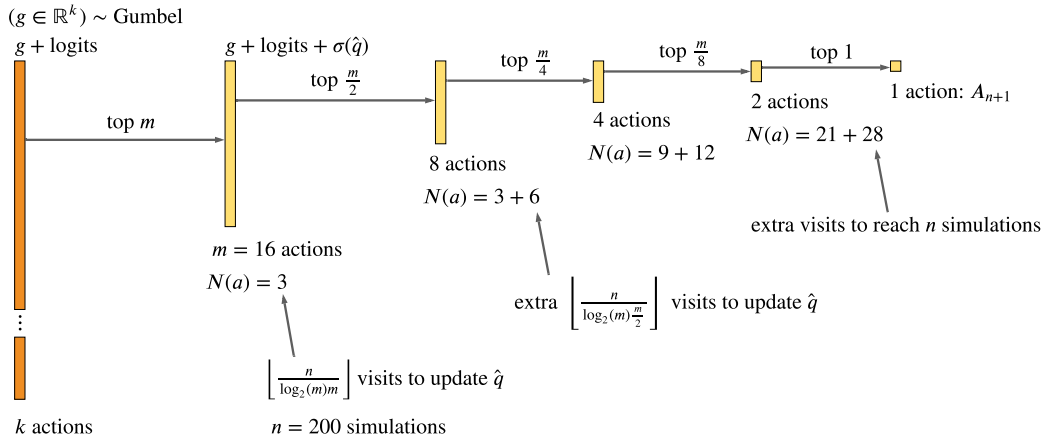


Figure 5.1: The number of considered actions and their visit counts $N(a)$ when using Sequential Halving with Gumbel on a k-armed stochastic bandit. The search uses $n = 200$ simulations and starts by sampling $m = 16$ actions without replacement. Sequential Halving divides the budget of n simulations equally to $\log_2(m)$ phases. In each phase, all considered actions are visited equally often. After each phase, one half of the actions are rejected. From the original k actions, only the best actions will remain.

For a concrete instantiation of σ, we use

$$\sigma(\hat{q}(a)) = (c_{\text{visit}} + \max_b N(b))c_{\text{scale}}\hat{q}(a), \tag{5.8}$$

where $\max_b N(b)$ is the visit count of the most-visited action. The transformation progressively increases the scale for $\hat{q}(a)$ and reduces the effect of the prior policy. This scale is inspired by the policy updates in MPO (Abdolmaleki et al., 2018; Vieillard et al., 2020). The finite scale for the Q-values provides regularized policy optimization and puts into effect the prior knowledge contained in the logits. Experimentally, $c_{\text{visit}} = 50$, $c_{\text{scale}} = 1.0$ allowed us to use the same hyperparameters even if changing the number of simulations.

5.4 Learning an improved policy

After the search, we have A_{n+1} from a (potentially) improved policy. Like AlphaZero, we would like to distill the improved policy to the policy network. One possibility is to train the policy network π to predict the A_{n+1}. That defines a simple policy loss:

$$L_{\text{simple}}(\pi) = -\log \pi(A_{n+1}). \tag{5.9}$$

Using completed Q-values. We will explain a different way to train the policy network, by extracting more knowledge from the search. Beside A_{n+1}, the search also gives us $q(a)$ (or its approximation) for the visited actions. We can construct an improved policy by first completing the vector of Q-values:

$$\text{completedQ}(a) = \begin{cases} q(a) & \text{if } N(a) > 0 \\ v_\pi, & \text{otherwise,} \end{cases} \tag{5.10}$$

where the unknown Q-values of the unvisited actions are replaced by $v_\pi = \sum_a \pi(a)q(a)$. While in practice we do not have the exact v_π, we have instead its approximation $\hat{v}_\pi$ from a value network. Even when training on off-policy data, we

devised a helpful v_π approximation (Appendix C.3).

With the completed Q-values, a new improved policy is constructed by

$$\pi' = \text{softmax}(\text{logits} + \sigma(\text{completedQ})), \tag{5.11}$$

where σ is a monotonically increasing transformation. We provide a proof of a policy improvement in Appendix C.2. Intuitively, the completed Q-values give zero advantage to the unvisited actions.

After constructing the new improved policy π', we can distill it to the policy network π:

$$L_{\text{completed}}(\pi) = \text{KL}(\pi', \pi). \tag{5.12}$$

This loss trains all actions, not only the action A_{n+1}. Later, we will investigate the effect of the loss in Figure 5.3a.

5.5 Planning at non-root nodes

To design an action selection for the non-root nodes of a search tree, we take inspiration from Grill et al. (2020). That allows us to interpret MCTS as regularized policy optimization. At a non-root node, we construct an improved policy π' by using the completed Q-values (Equation 5.11).

To select an action at the non-root node, one possibility is to sample the action from π'. However, sampling at non-root nodes adds unwanted variance to the estimated Q-values. Instead, we can design a deterministic action selection with the smallest mean-squared-error between the π' probabilities and the produced normalized visit counts. Such an action selection would select

$$\arg\min_a \sum_b \left(\pi'(b) - \underbrace{\frac{N(b) + \mathbb{I}\{a=b\}}{1 + \sum_c N(c)}}_{\text{Normalized visit counts, if taking } a.} \right)^2, \tag{5.13}$$

where the indicator $\mathbb{I}\{a=b\}$ is 1 if $a=b$, and zero otherwise. After a bit of algebra

(Appendix C.4), we obtain a simpler, more efficient expression:

$$\arg\max_a \left(\pi'(a) - \frac{N(a)}{1+\sum_b N(b)} \right). \tag{5.14}$$

This deterministic action selection selects the actions proportionally to π' and avoids an extra variance. We recommend a deterministic action selection only for non-root nodes. At the root node, the Gumbel noise is helpful for trying different actions in different episodes, while ensuring an improved expected value.

5.6 Related work

Rosin (2011) introduced the bandit with a predictor and designed PUCB ("Predictor + UCB") for cumulative regret minimization. AlphaGo (Silver et al., 2016), AlphaGo Zero (Silver et al., 2017a), AlphaZero (Silver et al., 2018), and MuZero (Schrittwieser et al., 2020) used a deep policy network as the predictor in a variant of the PUCB algorithm. Bertsekas (2019, 2021, 2022) provides an in-depth discussion of policy iteration, policy improvement, and their connection to rollout.

If not using a predictor, UCT (Kocsis and Szepesvári, 2006) would need to visit each action before being able to compare them. Rapid Action Value Estimation (Gelly and Silver, 2011) then helps to form rough estimates of the action values by aggregating statistics from all future states. Gelly and Silver (2011) also initialized the action value estimates with a heuristic evaluation function. The best heuristic used a learned linear network. Hamrick et al. (2020) later extended it to a deep Q-network.

Cazenave (2014) and Pepels et al. (2014) applied Sequential Halving to MCTS. Fabiano and Cazenave (2021) introduced Sequential Halving Using Scores. The 'scores' can be any prior offset to the Q-values. The way to obtain scores from a policy network was left as an open problem. We can now view the $g+\text{logits}$ as special scores.

MCTS is related to regularized policy optimization. Grill et al. (2020) analyzed AlphaZero tree search and discovered that AlphaZero approximates a regularized policy optimization. The approximation error is large if using a small number of

simulations. To avoid the approximation error, Grill et al. (2020) used a regularized policy optimization directly inside the tree search. In the setting without a predictor, Xiao et al. (2019) compared UCT to a new MCTS with an entropy regularizer. Dam et al. (2021) generalized it to relative entropy and Tsallis entropy. Regularized policy optimization is helpful when working with approximate Q-values (Vieillard et al., 2020) or when doing an approximate policy iteration (Kakade and Langford, 2002; Schulman et al., 2015).

TreeQN (Farquhar et al., 2018) uses a breadth-first search inside a network architecture. The network can do a lot of computation before producing a Q-value. To reduce the computation demands, Dynamic Planning Networks (Tasfi and Capretz, 2018) extended TreeQN to sample only some actions. To approximate the gradient, Dynamic Planning Networks use Gumbel-Softmax (Maddison et al., 2017; Jang et al., 2017). Although we use Gumbel variables, we do not employ approximate gradients from Gumbel-Softmax. We use the Gumbel-Top-k trick to construct efficient planning with a provable policy improvement.

For sampling without replacement, the unordered set estimator by Kool et al. (2020) provides an elegant, unbiased estimate of a gradient. However, gradient descent needs multiple steps to reach the solution of a regularized policy optimization problem (Tomar et al., 2020). Furthermore, the exact computation of the unordered set estimator requires $O(2^m)$ operations, which would be prohibitively expensive for $m = 16$. Practical time complexity can be achieved by using an importance-weighted estimator (Vieira, 2017; Nauman and Den Hengst, 2020).

Continuous actions can be supported by sampling k actions with replacement and then using the sampled actions as discrete actions with uniform logits for the rest of the search. This was done in Sampled MuZero (Hubert et al., 2021). Similarly, Critic Weighted Policy (Wang et al., 2020) uses sampling with replacement.

5.7 Experiments

In the experiments, we compare AlphaZero or MuZero to the proposed planning with Gumbel and other alternatives:

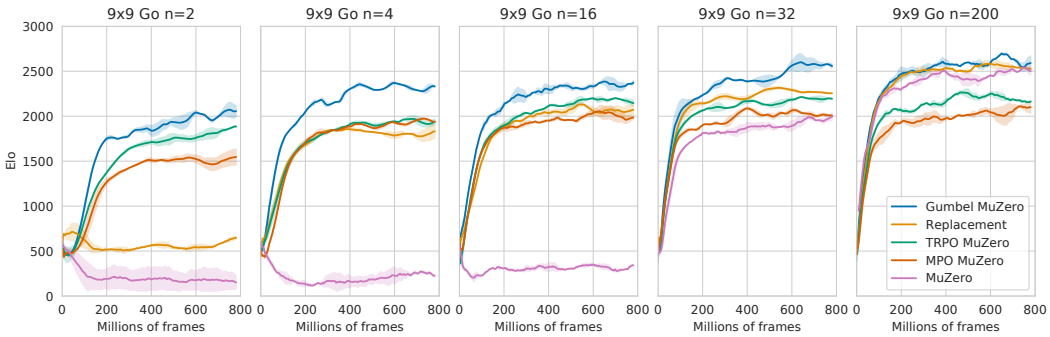


Figure 5.2: Elo on 9x9 Go, when training with $n \in \{2, 4, 16, 32, 200\}$ simulations. Evaluation uses 800 simulations. Shades denote standard errors from 2 seeds.

MuZero: The newest version of MuZero (Schrittwieser et al., 2021), with ResNet v2 style pre-activation residual blocks (He et al., 2016) and the Adam optimizer (Kingma and Ba, 2014). **Gumbel MuZero:** MuZero with the modified root of the search tree to use Sequential Halving with Gumbel. The policy loss uses the completed Q-values (Equation 5.12). **Gumbel MuZero sampled with replacement (Replacement):** An ablation to Gumbel MuZero by sampling m actions with replacement, as in Sampled MuZero (Hubert et al., 2021). **TRPO MuZero:** MuZero with modified learning, acting, and the root of the search tree to use the regularized policy optimization with the TRPO regularizer $\mathrm{KL}(\pi, \pi_{\text{new}})$ (Schulman et al., 2015; Grill et al., 2020). **MPO MuZero:** TRPO MuZero but with the MPO regularizer $\mathrm{KL}(\pi_{\text{new}}, \pi)$ (Abdolmaleki et al., 2018; Grill et al., 2020). **Full Gumbel MuZero:** Gumbel MuZero with a principled action selection also for the non-root search nodes (Section 5.5). In the plots, we will show Full Gumbel MuZero only if it produces results significantly different from Gumbel MuZero.

We conducted the experiments on Go, chess, and Atari. We present the main results here and we report additional ablations and experimental details in Appendix C.5.

5.7.1 9x9 Go

On Go, we use Elo to compare MuZero and other agents. While an agent trains by self-play, its Elo is computed by evaluation versus reference opponents. One of the opponents is Pachi (Baudiš and Gailly, 2011) with 10k simulations per move. We

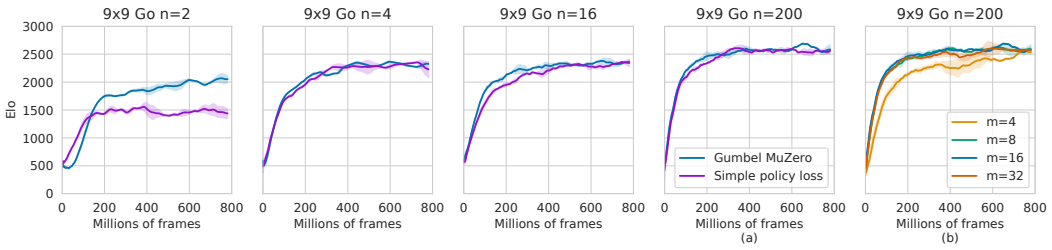


Figure 5.3: Gumbel MuZero ablations on 9x9 Go. **(a)** Policy loss ablations, when training with $n \in \{2, 4, 16, 200\}$ simulations. Gumbel MuZero uses the policy loss with completed Q-values. **(b)** Sensitivity to the number of sampled actions. Gumbel MuZero samples m actions without replacement.

anchored the Elo to have this Pachi at 1000 Elo. For example, a difference of 500 Elo corresponds to a 95% win probability for the player with the higher Elo.[2]

In Figure 5.2, we investigate the impact of the number of simulations on the obtained Elo. When training an agent by self-play, the agent uses n simulations per move. In the five plots, the n varies from 2 to 200. In evaluation, we allow all agents to use 800 simulations. The speed of the evaluation does not affect the speed of training. In the 9x9 Go results, MuZero fails to learn from 16 or fewer simulations. Strikingly, Gumbel MuZero learns reliably even with 2 simulations.

In Figure 5.3a, we compare the simple policy loss (Equation 5.9) and the policy loss with the completed Q-values (Equation 5.12). The simple policy loss would be enough for many applications. We used the completed Q-values also in TRPO MuZero and MPO MuZero. Without the completed Q-values, TRPO MuZero and MPO MuZero would fail to produce a policy improvement.

In Figure 5.3b, we study Gumbel MuZero's sensitivity to the number of sampled actions. When sampling $m = 4$ actions without replacement, the simulation budget is spent on the small number of actions. The learning was then slower. In all other Go experiments, we sample $m = \min(n, 16)$ actions without replacement.

5.7.2 Large-scale 19x19 Go and chess

In Figure 5.4a, we demonstrate that Gumbel MuZero is not worse than MuZero on 19x19 Go. MuZero is excellent on 19x19 Go and Gumbel MuZero reaches or exceeds its performance. The Elo is still anchored to have Pachi at 1000 Elo.

[2]The corresponding win probability is $\frac{1}{1+10^{-\frac{EloDifference}{400}}}$ (Elo, 1978).

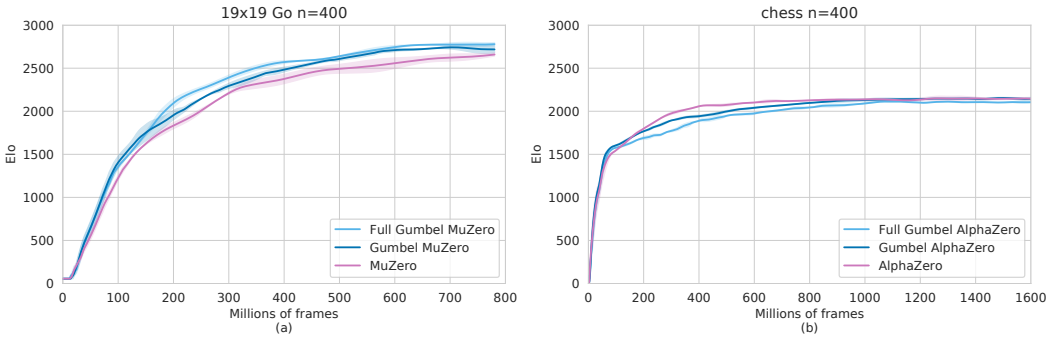


Figure 5.4: Large-scale experiments with $n = 400$ simulations per move. **(a)** Elo on 19x19 Go, when training MuZero. **(b)** Elo on chess, when training AlphaZero.

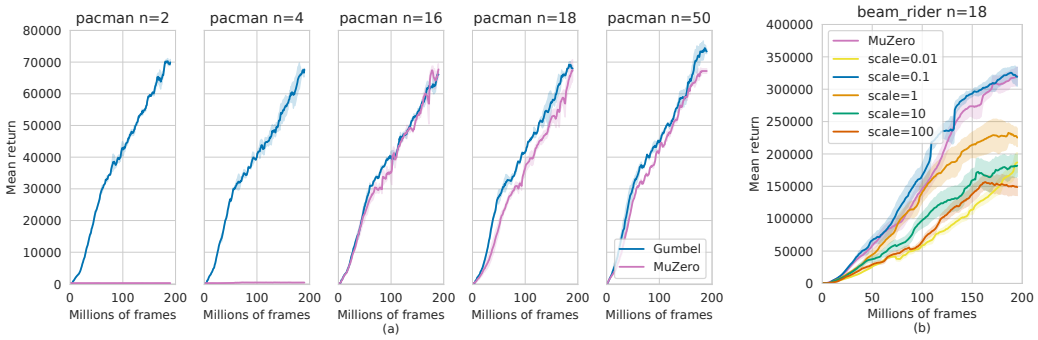


Figure 5.5: Atari results. **(a)** Mean return on `ms_pacman`, when training Gumbel MuZero and MuZero with $n \in \{2, 4, 16, 18, 50\}$ simulations. MuZero fails to learn from 4 or fewer simulations. **(b)** Mean return on `beam_rider` for Gumbel MuZero with $c_{\text{scale}} \in \{0.01, 0.1, 1, 10, 100\}$, compared to MuZero with $n = 50$ simulations. Shades denote standard errors from 10 seeds.

Similarly, in Figure 5.4b we show Gumbel AlphaZero performance on chess. We train AlphaZero on chess, because AlphaZero learns faster than MuZero on chess. On Go, MuZero learns faster than AlphaZero.

5.7.3 Atari

Our last set of experiments is on Atari. We use the Arcade Learning Environment (Bellemare et al., 2013) with sticky actions (Machado et al., 2018). The network sizes and hyperparameters match the MuZero setup by Schrittwieser et al. (2021). On Atari, MuZero does not use more simulations at evaluation. The reported score is the mean return from the last 200 training episodes. MuZero with $n = 50$ simulations works well on Atari and holds the state of the art.

In Figure 5.5a we show the obtained mean return on `ms_pacman`. Gumbel

MuZero again learns reliably even with $n = 2$ simulations. Atari has only 18 actions, so we sample $m = \min(n, 18)$ actions without replacement. In the experiments with $n \leq 18$, Gumbel MuZero selects A_{n+1} from the n visited actions, without any Sequential Halving. This confirms that planning with Gumbel is the key ingredient responsible for the policy improvement from a small number of simulations.

Atari is challenging, because different games can have very different reward scales. MuZero normalizes the Q-values by dividing them by $\max(\hat{v}_\pi, \max_a \hat{q}(a)) - \min(\hat{v}_\pi, \min_a \hat{q}(a))$ found inside the tree search (Schrittwieser et al., 2020). A normalized advantage is then in $[-1, 1]$. For Gumbel MuZero, we use the same normalization and we scale the normalized Q-values by $c_{\text{visit}} = 50$ and $c_{\text{scale}} = 0.1$. A scaled normalized advantage is then approximately in $[-5, 5]$. Thanks to the bounded advantage, Gumbel MuZero has a bounded total variation distance between π and π' (Muesli's Theorem 4.4.1).

In Figure 5.5b, we use `beam_rider` as an example of a partially observable game and we study the importance of the prior knowledge contained in the logits. Gumbel MuZero selects an action based on $g(a) + \text{logits}(a) + (c_{\text{visit}} + \max_b N(b))c_{\text{scale}}\hat{q}(a)$ (Equation 5.8). If c_{scale} is large, Gumbel MuZero focuses on $\hat{q}(a)$ and neglects the logits. Indeed, Gumbel MuZero performance is worse on `beam_rider` if using large c_{scale}. This indicates that $g(a) + \text{logits}(a)$ are helpful in the argmax.

5.8 Conclusion

We redesigned AlphaZero tree search. With the principle of policy improvement, we replaced five heuristic mechanisms in AlphaZero. On Go, chess, and Atari, we validated that Gumbel MuZero and Gumbel AlphaZero keep improving, even when learning from two simulations. On top of that, Gumbel MuZero provides a principled way to achieve state-of-the-art results. We hope that future research will benefit from the clean theoretical foundation, the faster experimentation with a small number of simulations, and the released open-source code.[3]

[3] `https://github.com/deepmind/mctx`

Chapter 6

General Conclusions

We will end with some practical conclusions.

In Chapter 3, we have seen that MuZero is not suitable for stochastic environments. If you want to plan with a learned model on a stochastic environment, design a causally correct model with a sufficient partial view. Or use the already tested Stochastic MuZero (Antonoglou et al., 2022). In general, be careful when training a model on a data distribution and then using the model on data from a different distribution. The causality literature provides many alarming counterexamples.

In Chapter 4, we have noticed multiple things: 1) Model learning is helpful as an auxiliary loss. 2) Deep search is helpful on Go, but deep search is not needed to achieve the current state-of-the-art results on Atari. 3) The Muesli policy update is helpful especially if using a smaller network with inaccurate Q-values. 4) Regularized policy optimization is not needed for tabular methods; regularized policy optimization becomes helpful on off-policy data or with approximate Q-values. 5) Do not regularize the policy entropy; use a better regularizer. For example, use the Muesli regularizer if having Q-values, or the Direct MPO regularizer if having only state-values.

In Chapter 5, we have seen another MuZero weakness: MuZero and AlphaZero can fail to produce a policy improvement if the number of simulations is smaller than the number of actions. The proposed Gumbel MuZero and Gumbel AlphaZero fix that. These algorithms also avoid the problem-dependent temperature and Dirichlet noise hyperparameters. On stochastic environments, you can

combine Gumbel MuZero with Stochastic MuZero (Antonoglou et al., 2022).

If you want to do planning, hear the Deep Learning Hypothesis by Ilya Sutskever: "*Anything a human can do in 0.1 seconds, a big 10-layer neural network can do, too!*" So, if your environment requires thinking for N seconds, consider using a neural network with $100N$ layers, or a recurrent network, or a deep search with $100N$ sequential steps.

Appendix A

Appendix to Causally Correct Partial Models for RL

Content

A.1 Backdoor and frontdoor adjustment formulas

Starting from a data-generation process of the form illustrated in Figure 3.3b, $p(x,y,u) = p(u)p(x|u)p(y|x,u)$, we can use the do-operator to compute $p(y|\text{do}(x)) = \int p(u)p(y|x,u)du$.

Without assuming any extra structure in $p(x|u)$ or in $p(y|x,u)$ it is not possible to compute $p(y|\text{do}(x))$ from the knowledge of the joint $p(x,y)$ alone.

If there was a variable z blocking all the effects of u on x, as illustrated in

Figure 3.3d, then $p(y|\text{do}(x))$ can be derived as follows:

$$\begin{aligned}
\text{Joint density} \quad & p(u)p(z|u)p(x|z)p(y|x,u) && \text{(A.1)}\\
\text{Intervention} \quad & p(x|z) \to \psi(x) && \text{(A.2)}\\
\text{Joint after intervention} \quad & p(u)p(z|u)\psi(x)p(y|x,u) && \text{(A.3)}\\
\text{Conditioning the new joint} \quad & p(y|\text{do}(x)) = \frac{\int p(u)p(z|u)\psi(x)p(y|x,u)dudz}{\psi(x)} \\
& = \int p(z)p(y|x,z)dz \\
& = \mathbb{E}_{p(z)}[p(y|x,z)], && \text{(A.4)}
\end{aligned}$$

where we used the formula

$$\begin{aligned}
\int p(u)p(z|u)p(y|x,u)du &= p(z)\int p(u|z)p(y|x,u)du && \text{(A.5)}\\
&= p(z)p(y|x,z). && \text{(A.6)}
\end{aligned}$$

If instead of just fixing the value of x, we perform a more general intervention $p(x|z) \to \psi(x|z)$, then $p_{\text{do}(\psi(x|z))}(y)$ can be derived as follows:

$$\begin{aligned}
\text{Joint density} \quad & p(u)p(z|u)p(x|z)p(y|x,u) && \text{(A.7)}\\
\text{Intervention} \quad & p(x|z) \to \psi(x|z) && \text{(A.8)}\\
\text{Joint after intervention} \quad & p(u)p(z|u)\psi(x|z)p(y|x,u) && \text{(A.9)}\\
\text{New marginal} \quad & p_{\text{do}(\psi(x|z))}(y) = \int p(u)p(z|u)\psi(x|z)p(y|x,u)dudzdx \\
& = \int p(z)\psi(x|z)p(y|x,z)dzdx \\
& = \mathbb{E}_{p(z)\psi(x|z)}[p(y|x,z)]. && \text{(A.10)}
\end{aligned}$$

Applying the same reasoning to the graph shown in Figure 3.3e, we obtain the formula

$$p(y|\text{do}(x)) = \mathbb{E}_{p(z|x)}[p(y|\text{do}(z))] = \mathbb{E}_{p(z|x)p(x')}[p(y|x',z)], \qquad \text{(A.11)}$$

where $p(z|x)$, $p(x')$ and $p(y|x',z)$ can be directly measured from the available (x,y,z) data. This formula holds as long as $p(z|x) > 0, \forall x,z$ and it is a simple instance of *frontdoor adjustment* (Pearl et al., 2016).

A.2 Tree search experiments

We bootstrap the MCTS search from a learned value function and use pUCT to select the simulation actions, as done in MuZero (Schrittwieser et al., 2020). Additionally, we enhance the MCTS with chance nodes (Browne et al., 2012). The value of a chance node is the expected value of its children, based on the learned $\hat{p}(z_t|h_t)$ probabilities. The chance nodes do not increase the number of network evaluations per a simulation, because if a child of the chance node is not expanded yet, we bootstrap from the child value.

When using expectimax or MCTS with chance nodes, we use the intended actions z_t as the chance outcomes. During a simulation, we generate the chance outcomes from the learned $\hat{p}(z_t|h_t)$ model. Conveniently, we are able to reuse the MuZero policy network as the $\hat{p}(z_t|h_t)$ model because the policy network is trained to model the intended actions. The deterministic non-causal partial model (NCPM) simply ignores the outcomes of the chance nodes. We do not recommend using the non-causal partial model together with MCTS with chance nodes. We still see that the MCTS with chance nodes helps the non-causal model, because it reduces the search depth and the agent bootstraps more from the correct $\hat{V}(s_0,a_0)$.

The best-performing hyperparameters were found by trying multiples of 3 for the learning rate, the probability of exploration and the policy cost weight. The final used hyperparameters are listed in Table A.1. The pUCT hyperparameters from MuZero (Schrittwieser et al., 2020) worked well. Each reported experiment was run with at least 8 independent random seeds. The shaded area in the plots indicates the standard error.

A.2.1 Details of experiments on MDPs

In Figure A.1, we see the results for the different models, when using expectimax. The models with clustered probabilities and clustered observations approxi-

Table A.1: Hyperparameters for the MDP and MiniPacman experiments.

Hyperparameter	Description	Value
μ	Learning rate	0.0003
β_1	Adam β_1	0.9
β_2	Adam β_2	0.999
`batch_size`	Mini-batch size	512
`max_depth`	Expectimax search depth	3
`num_simulations`	Number of MCTS simulations	50
`buffer_size`	Replay buffer size	500000
ε	Probability of exploration	0.01
c_{reward}	Reward cost weight	1.0
c_{value}	Value cost weight	1.0
c_{policy}	Policy cost weight	300.0
L_o	Overshoot Length	5
L_u	Length of n-step returns	10
γ	Discount factor	0.995

Model	Search depth 1	2	3
Non-causal	✓	✓	X
Intended action	✓	✓	✓
Clustered probs	✓	✓	✓
Clustered obs	✓	✓	✓

Figure A.1: Models solving the AvoidFuzzyBear MDP with expectimax. The deterministic non-causal model misled the agent when using search depth 3 or higher.

mate modeling of the probabilities or observations. These models are described in Appendix A.2.3.

A.2.2 Details of experiments on MiniPacman

We use the same stochastic version of MiniPacman as released by Guez et al. (2019). To make the task harder, we included two ghosts from the start of the game. We also increase the number of ghosts by one after solving two levels, four levels, or other multiples of 2. The game options are in Table A.2. To avoid having to stack frames, we draw the ghost movement direction to a pixel before the ghost. This makes the environment more Markovian.

The pretrained policy was trained by expectimax with search depth 1. This

Table A.2: The configuration used for the MiniPacman environments.

Argument	Description	Value
`frame_cap`	The maximum number of steps in an episode.	3000
`mode`	The game mode.	'regular'
`npills`	The number of power pills.	2
`nghosts`	The number of ghosts at the start of the game.	2
`ghost_speed_init`	The ghost probability of moving.	0.5
`ghost_speed_increase`	An increase to the ghost speed after a level.	0

is similar to Q-learning, except that the network consists of a reward model and a value network. This forms a strong baseline agent. Understandably, when training a model on data from the pretrained policy, expectimax performed worse than when training on on-policy data. The expectimax trained on the data from the pretrained policy was not able to test the proposed action in the real environment.

Replay with resampling. To improve data efficiency, we use a shuffling replay buffer and replay each trajectory four times. The trajectory does not have to store the used z_t. We store the used $m(z_t|s_t)$ distribution and resample the z_t from a posterior when replaying the trajectory. The posterior is:

$$p(z_t|s_t,a_t) = \frac{m(z_t|s_t)\pi(a_t|z_t)}{\sum_{z'_t} m(z'_t|s_t)\pi(a_t|z'_t)} \tag{A.12}$$

where $\pi(a_t|z_t)$ is the ε-exploration distribution.

A.2.3 Models trained by clustering

When using a tree search, we want to have a small branching factor at the chance nodes. A good z_t variable would be discrete with a small number of categories. This is satisfied if the z_t is the intended action and the number of the possible actions is small. We do not have such compact discrete z_t if using as z_t the observation, the policy probabilities, or some other modeled layer. Here, we will present a model that approximates such causal partial models. The idea is to cluster the modeled layers and use only the cluster index as z_t. The cluster index is discrete and we can control the branching factor by choosing the number of clusters.

Concretely, let's call the modeled layer x_t. We will model the layer with a mix-

ture of components. The mixture gives us a discrete latent variable z_t to represent the component index. To train the mixture, we use a clustering loss to train only the best component to model the x_t, given h_t and z_t:

$$L_{\text{clustering}} = \min_{z_t}(-\beta_{\text{clustering}} \log \hat{p}(z_t|h_t) - \log \hat{p}(x_t|h_t, z_t)) \tag{A.13}$$

where $\hat{p}(z_t|h_t)$ is a model of the categorical component index and $\beta_{\text{clustering}} \in (0,1)$ is a hyperparameter to encourage moving the information bits to the latent z_t. During training, we use the index of the best component as the inferred z_t. In theory, a better inference can be obtained by smoothing.

In contrast to training by maximum likelihood, the clustering loss uses just the needed number of the mixture components. This helps to reduce the branching factor in a search.

In general, the cluster index is not guaranteed to be sufficient as a backdoor, if the reconstruction loss $-\log \hat{p}(x_t|h_t, z_t)$ is not zero. For example, if x_t is the next observation, the number of mixture components may need to be unrealistically large, if the observation can contain many distractors.

A.3 Value-iteration analysis on MDPs

We derive the following two model-based evaluation metrics for any episodic MDP environment:

- $V^*_{\text{NCPM}(\pi)}(s_0)$: the optimal value computed with the deterministic non-causal model, after the model is trained with training data from policy π, starting from state s_0.

- $V^*_{\text{CPM}(\pi)}(s_0)$: the optimal value computed with the causal partial model, after the model is trained with training data from policy π, starting from state s_0.

The theoretical analysis of the MDP does not use empirically trained models from the policy data, but rather assumes that the models learned perfectly the mean of their targets. That means that the non-causal (MuZero) reward model

$\hat{r}_k(s_t, a_{<t+k})$ predicts the expected reward $\mathbb{E}[R_{t+k}|s_t, a_{<t+k}]$, and that the causally correct reward model $\hat{r}_k(s_t, a_t, z_{<t+k}, a_{<t+k})$ predicts $\mathbb{E}[R_{t+k}|s_t, a_t, z_{<t+k}, a_{<t+k}]$.

Computation of $V^*_{\mathbf{NCPM}(\pi)}$: For the deterministic non-causal model,

$$\begin{aligned} V^*_{\mathrm{NCPM}(\pi)}(s_0) &= \max_{a_0,\ldots,a_k} \sum_{i=0}^{k} \mathbb{E}_{s_i}\left[r_{i+1}(s_i, a_i) \mid s_0, a_0, a_1, \ldots, a_i\right] \\ &= \max_{a_0,\ldots,a_k} \sum_{i=0}^{k} \sum_{s_i} p(s_i \mid s_0, a_0, a_1, \ldots, a_i) r_{i+1}(s_i, a_i). \end{aligned}$$

Notice that the probability of s_i is affected by a_i here because the network gets a_i as an input when predicting the r_{i+1}. This will introduce the non-causal bias. The network implements the expectation implicitly by learning the mean of the reward seen in the training data. We can compute the expectation exactly if we know the MDP. The $p(s_i \mid s_0, a_0, \ldots, a_i)$ can be computed recursively in two steps as:

$$p(s_i \mid s_0, a_0, \ldots, a_i) = \frac{p(s_i \mid s_0, a_0, \ldots, a_{i-1})\pi(a_i \mid s_i)}{\sum_{s'_i} p(s'_i \mid s_0, a_0, \ldots, a_{i-1})\pi(a_i \mid s'_i)}.$$

Here, we see the dependence of the learned model on the policy π. The remaining terms can be expressed as:

$$\begin{aligned} p(s_i \mid s_0, a_0, \ldots, a_{i-1}) &= \sum_{s_{i-1}} p(s_i, s_{i-1} \mid s_0, a_0, \ldots, a_{i-1}) \\ &= \sum_{s_{i-1}} p(s_{i-1} \mid s_0, a_0, \ldots, a_{i-1}) p(s_i \mid s_{i-1}, a_{i-1}). \end{aligned}$$

Denoting $p(s_i \mid s_0, a_0, \ldots, a_j)$ by $S_{i,j}$, we have the two-step recursion

$$\begin{aligned} S_{i,i} &= \frac{S_{i,i-1}\,\pi(a_i \mid s_i)}{\sum_{s'_i} S'_{i,i-1}\,\pi(a_i \mid s'_i)}, \\ S_{i,i-1} &= \sum_{s_{i-1}} S_{i-1,i-1}\, p(s_i \mid s_{i-1}, a_{i-1}) \end{aligned}$$

with $S_{1,0} = p(s_1 \mid s_0, a_0)$. We then compute $V^*_{\mathrm{NCPM}}(s_0)$ as $\max_{a_0,\ldots,a_k} \sum_{i=0}^{k} \sum_{s_i} S_{i,i} r_{i+1}(s_i, a_i)$.

Computation of $V^*_{\mathbf{CPM}(\pi)}$: For the causal partial model,

$$\begin{aligned} V^*_{\mathrm{CPM}(\pi)}(s_0) = &\max_{a_0} \sum_{z_1} p(z_1 \mid s_0, a_0) \max_{a_1} \sum_{z_2} p(z_2 \mid s_0, a_0, z_1, a_1) \cdots \\ &\max_{a_{k-1}} \sum_{z_k} p(z_k \mid s_0, a_0, z_1, a_1, \ldots, z_{k-1}, a_{k-1}) \\ &\max_{a_k} \sum_{i=0}^{k} \mathbb{E}[r_{i+1}(s_i, a_i) \mid s_0, a_0, z_1, a_1, \ldots, a_i)], \end{aligned}$$

where for any $i \in [1,k]$,

$$\begin{aligned} p(z_i \mid s_0, a_0, z_1, a_1, \ldots, z_{i-1}, a_{i-1}) &= \sum_{s_i} p(s_i, z_i \mid s_0, a_0, z_1, a_1, \ldots, z_{i-1}, a_{i-1}) \\ &= \sum_{s_i} p(s_i \mid s_0, a_0, z_1 \ldots, z_{i-1}, a_{i-1})\, \pi(z_i \mid s_i). \end{aligned}$$

Denoting $p(s_i \mid s_0, a_0, z_1 \ldots, z_{i-1}, a_{i-1})$ by Z_i, we have

$$\begin{aligned} Z_i &= \sum_{s_{i-1}} p(s_{i-1}, s_i \mid s_0, a_0, z_1 \ldots, z_{i-1}, a_{i-1}) \\ &= \sum_{s_{i-1}} p(s_i \mid s_{i-1}, a_{i-1}) p(s_{i-1} \mid s_0, a_0, z_1 \ldots, z_{i-1}, a_{i-1}) \\ &= \sum_{s_{i-1}} p(s_i \mid s_{i-1}, a_{i-1}) p(s_{i-1} \mid s_0, a_0, z_1 \ldots, z_{i-1}), \end{aligned}$$

where we used the fact that s_{i-1} is independent of a_{i-1}, given z_{i-1}. Furthermore,

$$\begin{aligned} p(s_{i-1} \mid s_0, a_0, z_1 \ldots, z_{i-1}) &= \frac{p(s_{i-1}, z_{i-1} \mid s_0, a_0, z_1 \ldots, a_{i-2})}{p(z_{i-1} \mid s_0, a_0, z_1 \ldots, a_{i-2})} \\ &= \frac{\pi(z_{i-1} \mid s_{i-1}) p(s_{i-1} \mid s_0, a_0, z_1 \ldots, z_{i-2}, a_{i-2})}{\sum_{s'_{i-1}} \pi(z_{i-1} \mid s'_{i-1}) p(s'_{i-1} \mid s_0, a_0, z_1 \ldots, z_{i-2}, a_{i-2})} \\ &= \frac{\pi(z_{i-1} \mid s_{i-1}) Z_{i-1}}{\sum_{s'_{i-1}} \pi(z_{i-1} \mid s'_{i-1}) Z'_{i-1}}. \end{aligned}$$

Therefore, we can compute Z_i recursively,

$$Z_i = \sum_{s_{i-1}} p(s_i \mid s_{i-1}, a_{i-1}) \frac{\pi(z_{i-1} \mid s_{i-1}) Z_{i-1}}{\sum_{s'_{i-1}} \pi(z_{i-1} \mid s'_{i-1}) Z'_{i-1}}$$

with $Z_1 = p(s_1 \mid s_0, a_0)$. The last term to compute in the definition of $V^*_{\mathrm{CPM}(\pi)}(s_0)$ is

$$\begin{aligned}
\sum_{i=0}^{k} \mathbb{E}[r_{i+1}(s_i, a_i) \mid s_0, a_0, z_1, a_1, \ldots, a_i)] &= \sum_{i=0}^{k} \mathbb{E}[r_{i+1}(s_i, a_i) \mid s_0, a_0, z_1, a_1, \ldots, z_i)] \\
&= \sum_{i=0}^{k} \sum_{s_i} p(s_i \mid s_0, a_0, z_1, a_1, \ldots, z_i) r_{i+1}(s_i, a_i) \\
&= \sum_{i=0}^{k} \sum_{s_i} \frac{\pi(z_i \mid s_i) Z_i}{\sum_{s'_i} \pi(z_i \mid s'_i) Z'_i} r_{i+1}(s_i, a_i).
\end{aligned}$$

Appendix B

Muesli Supplement

Content

B.1 Stochastic estimation details

In the policy-gradient term in Eq. 4.7, we clip the importance weight $\frac{\pi(A|s)}{\pi_b(A|s)}$ to be from $[0, 1]$. The importance weight clipping introduces a bias. To correct for it, we use β-LOO action-dependent baselines (Gruslys et al., 2018).

Although the β-LOO action-dependent baselines were not significant in the Muesli results, the β-LOO was helpful for the policy gradients with the TRPO penalty (Figure B.8).

B.2 The illustrative MDP example

Here we will analyze the values and the optimal policy for the MDP from Figure B.1 when using the identical state representation $\phi(s) = \varnothing$ in all states. With the state

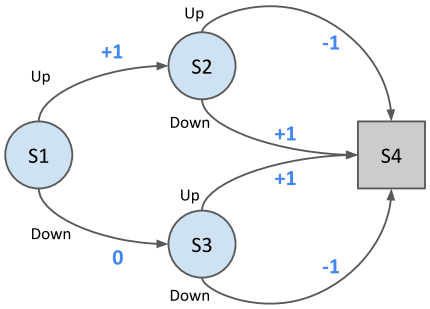


Figure B.1: The episodic MDP from Figure 4.2, reproduced here for easier reference. State 1 is the initial state. State 4 is terminal. The discount is 1.

representation $\phi(s)$, the policy is restricted to be the same in all states. Let's denote the probability of the *up* action by p.

Given the policy $p = \pi(up|\phi(s))$, the following are the values of the different states:

$$v_\pi(3) = p + (1-p)(-1) = 2p - 1 \tag{B.1}$$

$$v_\pi(2) = p \cdot (-1) + (1-p) = -2p + 1 \tag{B.2}$$

$$v_\pi(1) = p \cdot (1 + v_\pi(2)) + (1-p)v_\pi(3) \tag{B.3}$$

$$= -4p^2 + 5p - 1. \tag{B.4}$$

Finding the optimal policy. Our objective is to maximize the value of the initial state. That means maximizing $v_\pi(1)$. We can find the maximum by looking at the derivatives. The derivative of $v_\pi(1)$ with respect to the policy parameter is:

$$\frac{dv_\pi(1)}{dp} = -8p + 5. \tag{B.5}$$

The second derivative is negative, so the maximum of $v_\pi(1)$ is at the point where the first derivative is zero. We conclude that the maximum of $v_\pi(1)$ is at $p^* = \frac{5}{8}$.

Finding the action values of the optimal policy. We will now find the $q^*_\pi(\phi(s), up)$ and $q^*_\pi(\phi(s), down)$. The $q_\pi(\phi(s), a)$ is defined as the expected return

after the $\phi(s)$ when *doing* the action a (Singh et al., 1994):

$$q_\pi(\phi(s),a) = \sum_{s'} P_\pi(s'|\phi(s)) q_\pi(s',a), \tag{B.6}$$

where $P_\pi(s'|\phi(s))$ is the probability of being in the state s' when observing $\phi(s)$.

In our example, the Q-values are:

$$\begin{aligned} q_\pi(\phi(s),up) &= \frac{1}{2}(1+v_\pi(2)) + \frac{1}{2}p\cdot(-1) + \frac{1}{2}(1-p) && \text{(B.7)}\\ &= -2p + \frac{3}{2} && \text{(B.8)}\\ q_\pi(\phi(s),down) &= \frac{1}{2}v_\pi(3) + \frac{1}{2}p + \frac{1}{2}(1-p)(-1) && \text{(B.9)}\\ &= 2p - 1 && \text{(B.10)} \end{aligned}$$

We can now substitute the $p^* = \frac{5}{8}$ in for p to find the $q^*_\pi(\phi(s),up)$ and $q^*_\pi(\phi(s),down)$:

$$\begin{aligned} q^*_\pi(\phi(s),up) &= \frac{1}{4} && \text{(B.11)}\\ q^*_\pi(\phi(s),down) &= \frac{1}{4}. && \text{(B.12)} \end{aligned}$$

We see that these Q-values are the same and uninformative about the probabilities of the optimal (memory-less) stochastic policy. This generalizes to all environments: the optimal policy gives zero probability to all actions with lower Q-values. If the optimal policy $\pi^*(\cdot|\phi(s))$ at a given state representation gives non-zero probabilities to some actions, these actions must have the same Q-values $q^*_\pi(\phi(s),a)$.

Bootstrapping from $v_\pi(\phi(s))$ would be worse. We will find the $v_\pi(\phi(s))$, and we will show that bootstrapping from it would be misleading. In our example, the $v_\pi(\phi(s))$ is:

$$\begin{aligned} v_\pi(\phi(s)) &= \frac{1}{2}v_\pi(1) + \frac{1}{2}pv_\pi(2) + \frac{1}{2}(1-p)v_\pi(3) && \text{(B.13)}\\ &= -4p^2 + \frac{9}{2}p - 1. && \text{(B.14)} \end{aligned}$$

We can notice that $v_\pi(\phi(s))$ is different from $v_\pi(2)$ or $v_\pi(3)$. Estimating $q_\pi(\phi(s), up)$ by bootstrapping from $v_\pi(\phi(2))$ instead of $v_\pi(2)$ would be misleading. Here, it is better to estimate the Q-values based on Monte-Carlo returns.

B.3 The motivation behind Conservative Policy Iteration and TRPO

In this section, we will show that unregularized maximization of $\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{q}_{\pi_{\text{prior}}}(s,A)\right]$ on data from an older policy π_{prior} can produce a policy worse than π_{prior}. The size of the possible degradation will be related to the total variation distance between π and π_{prior}. The explanation is based on the proofs from the excellent book by Agarwal et al. (2020b).

As before, our objective is to maximize the expected value of the states from an initial state distribution μ:

$$J(\pi) = \mathbb{E}_{S\sim\mu}\left[v_\pi(S)\right]. \tag{B.15}$$

It will be helpful to define the discounted state visitation distribution $d_\pi(s)$ as:

$$d_\pi(s) = (1-\gamma)\mathbb{E}_{S_0\sim\mu}\left[\sum_{t=0}^{\infty}\gamma^t P(S_t = s|\pi, S_0)\right], \tag{B.16}$$

where $P(S_t = s|\pi, S_0)$ is the probability of S_t being s, if starting the episode from S_0 and following the policy π. The scaling by $(1-\gamma)$ ensures that $d_\pi(s)$ sums to one.

From the policy gradient theorem (Sutton et al., 2000) we know that the gradient of $J(\pi)$ with respect to the policy parameters is

$$\frac{\partial J}{\partial \theta} = \frac{1}{1-\gamma}\sum_s d_\pi(s)\sum_a \frac{\partial \pi(a|s)}{\partial \theta} q_\pi(s,a). \tag{B.17}$$

In practice, we often train on data from an older policy π_{prior}. Training on such

data maximizes a different function:

$$\text{TotalAdv}_{\text{prior}}(\pi) = \frac{1}{1-\gamma}\sum_s d_{\pi_{\text{prior}}}(s)\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a), \tag{B.18}$$

where $\text{adv}_{\pi_{\text{prior}}}(s,a) = q_{\pi_{\text{prior}}}(s,a) - v_{\pi_{\text{prior}}}(s)$ is an advantage. Notice that the states are sampled from $d_{\pi_{\text{prior}}}(s)$ and the policy is criticized by $\text{adv}_{\pi_{\text{prior}}}(s,a)$. This often happens in the practice when updating the policy multiple times in an episode, using a replay buffer, or bootstrapping from a network trained on past data.

While maximization of $\text{TotalAdv}_{\text{prior}}(\pi)$ is more practical, we will see that unregularized maximization of $\text{TotalAdv}_{\text{prior}}(\pi)$ does not guarantee an improvement in our objective J. The $J(\pi) - J(\pi_{\text{prior}})$ difference can even be negative if we are not careful.

Kakade and Langford (2002) stated a useful lemma for the performance difference:

Lemma B.3.1 (The performance difference lemma)**.** *For all policies* π, π_{prior},

$$J(\pi) - J(\pi_{\text{prior}}) = \frac{1}{1-\gamma}\sum_s d_\pi(s)\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a). \tag{B.19}$$

We would like the $J(\pi) - J(\pi_{\text{prior}})$ to be positive. We can express the performance difference as $\text{TotalAdv}_{\text{prior}}(\pi)$ plus an extra term:

$$\begin{aligned}
J(\pi) - J(\pi_{\text{prior}}) =& \text{TotalAdv}_{\text{prior}}(\pi) \\
&- \text{TotalAdv}_{\text{prior}}(\pi) + \frac{1}{1-\gamma}\sum_s d_\pi(s)\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a) \\
=& \text{TotalAdv}_{\text{prior}}(\pi) \\
&+ \frac{1}{1-\gamma}\sum_s (d_\pi(s) - d_{\pi_{\text{prior}}}(s))\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a) \\
=& \text{TotalAdv}_{\text{prior}}(\pi) \\
&+ \frac{1}{1-\gamma}\sum_s (d_\pi(s) - d_{\pi_{\text{prior}}}(s))\sum_a (\pi(a|s) - \pi_{\text{prior}}(a|s))\,\text{adv}_{\pi_{\text{prior}}}(s,a).
\end{aligned} \tag{B.20}$$

To get a positive $J(\pi) - J(\pi_{\text{prior}})$ performance difference, it is not enough to maximize $\text{TotalAdv}_{\text{prior}}(\pi)$. We also need to ensure that the second term in (B.20) will not degrade the performance. The impact of the second term can be kept small by keeping the total variation distance between π and π_{prior} small.

For example, the performance can degrade if π is not trained at a state and that state gets a higher $d_\pi(s)$ probability. The performance can also degrade if a stochastic policy is needed and the $\text{adv}_{\pi_{\text{prior}}}$ advantages are for an older policy. The π would become deterministic if maximizing $\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a)$ without any regularization.

B.3.1 Performance difference lower bound

We will express a bound of the performance difference as a function of the total variation between π and π_{prior}. Starting from Eq. B.20, we can derive the TRPO lower bound for the performance difference. Let α be the maximum total variation distance between π and π_{prior}:

$$\alpha = \max_s \frac{1}{2} \sum_a |\pi(a|s) - \pi_{\text{prior}}(a|s)|. \tag{B.21}$$

The $\|d_\pi - d_{\pi_{\text{prior}}}\|_1$ is then bounded (see Agarwal et al., 2020b, "Similar policies imply similar state visitations"):

$$\|d_\pi - d_{\pi_{\text{prior}}}\|_1 \leq \frac{2\alpha\gamma}{1-\gamma}. \tag{B.22}$$

Finally, by plugging the bounds into Eq. B.20, we can construct the lower bound for the performance difference:

$$J(\pi) - J(\pi_{\text{prior}}) \geq \text{TotalAdv}_{\text{prior}}(\pi) - \frac{4\alpha^2\gamma\varepsilon_{\max}}{(1-\gamma)^2}, \tag{B.23}$$

where $\varepsilon_{\max} = \max_{s,a} |\,\text{adv}_{\pi_{\text{prior}}}(s,a)|$. The same bound was derived in TRPO (Schulman et al., 2015).

B.4 Proof of Maximum CMPO total variation distance

We will prove the following theorem: *For any clipping threshold $c > 0$, we have:*

$$\max_{\pi_{\text{prior}},\hat{\text{adv}},s} \mathrm{D}_{\text{TV}}(\pi_{\text{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s)) = \tanh(\frac{c}{2}). \tag{B.24}$$

Having 2 actions. We will first prove the theorem when the policy has 2 actions. To maximize the distance, the clipped advantages will be $-c$ and c. Let's denote the π_{prior} probabilities associated with these advantages as $1-p$ and p, respectively.

The total variation distance is then:

$$\mathrm{D}_{\text{TV}}(\pi_{\text{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s)) = \frac{p\exp(c)}{p\exp(c) + (1-p)\exp(-c)} - p. \tag{B.25}$$

We will maximize the distance with respect to the parameter $p \in [0,1]$.

The first derivative with respect to p is

$$\frac{d\,\mathrm{D}_{\text{TV}}(\pi_{\text{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s))}{dp} = \frac{1}{(p\exp(c) + (1-p)\exp(-c))^2} - 1. \tag{B.26}$$

The second derivative with respect to p is

$$\frac{d^2\,\mathrm{D}_{\text{TV}}(\pi_{\text{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s))}{dp^2} = -2(p\exp(c) + (1-p)\exp(-c))^{-3}(\exp(c) - \exp(-c)). \tag{B.27}$$

Because the second derivative is negative, the distance is a concave function of p. We will find the maximum at the point where the first derivative is zero. The solution is

$$p^* = \frac{1-\exp(-c)}{\exp(c) - \exp(-c)}. \tag{B.28}$$

At the found point p^*, the maximum total variation distance is

$$\max_p \mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\mathrm{prior}}(\cdot|s)) = \frac{\exp(c)-1}{\exp(c)+1} = \tanh(\frac{c}{2}). \tag{B.29}$$

This completes the proof when having 2 actions.

Having any number of actions. We will now prove the theorem when the policy has any number of actions. To maximize the distance, the clipped advantages will be $-c$ or c. Let's denote the *sum* of π_{prior} probabilities associated with these advantages as $1-p$ and p, respectively.

The total variation distance is again

$$\mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\mathrm{prior}}(\cdot|s)) = \frac{p\exp(c)}{p\exp(c)+(1-p)\exp(-c)} - p, \tag{B.30}$$

and the maximum distance is again $\tanh(\frac{c}{2})$.

We also verified the theorem predictions experimentally by using gradient ascent to maximize the total variation distance.

B.5 Extended related work

We used the desiderata to motivate the design of the policy update. We will use the desiderata again to discuss the related methods used to satisfy the desiderata. For a comprehensive overview of model-based reinforcement learning, we recommend the surveys by Moerland et al. (2020) and Hamrick (2019).

B.5.1 Observability and function approximation

1a) Support learning stochastic policies. The ability to learn a stochastic policy is one of the benefits of policy gradient methods.

1b) Leverage Monte-Carlo targets. Muesli uses multi-step returns to train the policy network and Q-values. MPO and MuZero need to train the Q-values before using the Q-values to train the policy.

B.5.2 Policy representation

2a) Support learning the optimal memory-less policy. Muesli represents the stochastic policy by the learned policy network. In principle, acting can be based on a combination of the policy network and the Q-values. For example, one possibility is to act with the π_{CMPO} policy. ACER (Wang et al., 2016) used similar acting based on π_{MPO}. Although we have not seen benefits from acting based on π_{CMPO} on Atari (Figure B.7), we have seen better results on Go with a deeper search at the evaluation time.

2b) Scale to (large) discrete action spaces. Muesli supports large actions spaces because the policy loss can be estimated by sampling. MCTS is less suitable for large action spaces. This was addressed by Grill et al. (2020), who brilliantly revealed MCTS as regularized policy optimization and designed a tree search based on MPO or a different regularized policy optimization. The resulting tree search was less affected by a small number of simulations. Muesli is based on this view of regularized policy optimization as an alternative to MCTS. In another approach, MuZero was recently extended to support sampled actions and continuous actions (Hubert et al., 2021). Most recently, Gumbel MuZero (Chapter 5) supports large actions spaces, even if using a small number of simulations.

2c) Scale to continuous action spaces. Although we used the same estimator of the policy loss for discrete and continuous actions, it would be possible to exploit the structure of the continuous policy. For example, the continuous policy can be represented by a normalizing flow (Papamakarios et al., 2019) to model the joint distribution of the multi-dimensional actions. Soft Actor-Critic (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) achieved great results on the MuJoCo tasks by obtaining the gradient with respect to the action from an ensemble of approximate Q-functions. The ensemble of Q-functions would probably improve Muesli results.

B.5.3 Robust learning

3a) Support off-policy and historical data. Muesli supports off-policy data thanks to the regularized policy optimization, Retrace (Munos et al., 2016), and policy gra-

dients with clipped importance weights (Gruslys et al., 2018). Many other methods deal with off-policy or offline data (Levine et al., 2020). Recently MuZero Reanalyse (Schrittwieser et al., 2021) achieved state-of-the-art results on an offline RL benchmark by training only on the offline data.

3b) Deal gracefully with inaccuracies in the values/model. Muesli does not trust fully the Q-values from the model. Muesli combines the Q-values with the prior policy to propose a new policy with a constrained total variation distance from the prior policy. Without the regularized policy optimization, the agent can be misled by an overestimated Q-value for a rarely taken action. Soft Actor-Critic (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) mitigate the overestimation by taking the minimum from a pair of Q-networks. In model-based reinforcement learning, an unrolled one-step model would struggle with compounding errors (Janner et al., 2019). VPN (Oh et al., 2017) and MuZero (Schrittwieser et al., 2020) avoid compounding errors by using multi-step predictions $P(R_{t+k+1}|s_t, a_t, a_{t+1}, \ldots, a_{t+k})$, not conditioned on previous model predictions. While VPN and MuZero avoid compounding errors, these models are not suitable for planning a sequence of actions in a stochastic environment. In the stochastic environment, the sequence of actions needs to depend on the occurred stochastic events, otherwise the planning is confounded and can overestimate the state value (Chapter 3). Other models conditioned on limited information from generated (latent) variables can face similar problems on stochastic environments (e.g., DreamerV2 (Hafner et al., 2020)). Muesli is suitable for stochastic environments because Muesli uses only one-step look-ahead. If combining Muesli with a deep search, we can use an adaptive search depth or a stochastic model sufficient for causally correct planning (Chapter 3). Another class of models deals with model errors by using the model as a part of the Q-network or policy network and trains the whole network end-to-end. These networks include VIN (Tamar et al., 2016), Predictron (Silver et al., 2017b), I2A (Racanière et al., 2017), IBP (Pascanu et al., 2017), TreeQN, ATreeC (Farquhar et al., 2018) (with scores in Table B.1), ACE (Zhang and Yao, 2019), UPN (Srinivas et al., 2018), and implicit planning with

DRC (Guez et al., 2019).

3c) Be robust to diverse reward scales. Muesli benefits from the normalized advantages and from the advantage clipping inside π_{CMPO}. Pop-Art (van Hasselt et al., 2016) addressed learning values across many orders of magnitude. On Atari, the score of the games vary from 21 on Pong to 1M on Atlantis. The non-linear transformation by Pohlen et al. (2018) is practically very helpful, although biased for stochastic returns. Only recently, an unbiased version of the transformation was added by Eszter Vértes to RLax.[1]

3d) Avoid problem-dependent hyperparameters. The normalized advantages were used before in PPO (Schulman et al., 2017). The maximum CMPO total variation (Theorem 4.4.1) helps to explain the success of such normalization. If the normalized advantages are from $[-c, c]$, they behave like advantages clipped to $[-c, c]$. Notice that the regularized policy optimization with the popular $-\text{H}[\pi]$ entropy regularizer is equivalent to MPO with uniform π_{prior} (because $-\text{H}[\pi] = \text{KL}(\pi, \pi_{\text{uniform}}) + \text{const.}$). As a simple modification, we recommend replacing the uniform prior with π_{prior} based on a target network. That leads to the model-free direct MPO with normalized advantages, outperforming vanilla policy gradients (compare Figure B.5 to Figure 4.1a).

B.5.4 Rich representation of knowledge

4a) Estimate values (variance reduction, bootstrapping). In Muesli, the learned values are helpful for bootstrapping Retrace returns, for computing the advantages, and for constructing the π_{CMPO}. Q-values can be also helpful inside a search, as demonstrated by Hamrick et al. (2020).

4b) Learn a model (representation, composability). Multiple works have demonstrated benefits from learning a model. Like VPN and MuZero, Gregor et al. (2019) learn a multi-step action-conditional model; they learn the distribution of observations instead of actions and rewards, and focus on the benefits of representation learning in model-free RL induced by model-learning; see also (Guo et al., 2018; Guo et al., 2020). Springenberg et al. (2020) study an algorithm similar to

[1] `https://github.com/deepmind/rlax/pull/78`

MuZero with an MPO-like learning signal and obtain strong results on MuJoCo tasks. Byravan et al. (2020) use a multi-step action model to derive a learning signal for policies on continuous-valued actions, leveraging the differentiability of the model and of the policy. Kaiser et al. (2019) show how to use a model for increasing data-efficiency on Atari (using an algorithm similar to Dyna (Sutton, 1990)), but see also van Hasselt et al. (2019) for the relation between parametric model and replay. Finally, Hamrick et al. (2021) investigate drivers of performance and generalization in MuZero-like algorithms.

Table B.1: The mean score from the last 100 episodes at 40M frames on games used by TreeQN and ATreeC. The agents differ along multiple dimensions.

	Alien	Amidar	Crazy Climber	Enduro	Frostbite	Krull	Ms. Pacman	Q*Bert	Seaquest
TreeQN-1	2321	1030	107983	800	2254	10836	3030	15688	9302
TreeQN-2	2497	1170	104932	825	581	11035	3277	15970	8241
ATreeC-1	3448	**1578**	102546	678	1035	8227	4866	25159	1734
ATreeC-2	2813	1566	110712	649	281	8134	4450	25459	2176
Muesli	**16218**	524	**143898**	**2344**	**10919**	**15195**	**19244**	**30937**	**142431**

B.6 Experimental details

B.6.1 Common parts

Network architecture. The large MuZero network is used only on the large-scale Atari experiments (Figure 4.1b) and on Go. In all other Atari and MuJoCo experiments the network architecture is based on the IMPALA architecture (Espeholt et al., 2018). Like the LASER agent (Schmitt et al., 2020), we increase the number of channels four times. Specifically, the numbers of channels are: (64, 128, 128, 64), followed by a fully connected layer and LSTM (Hochreiter and Schmidhuber, 1997) with 512 hidden units. This LSTM inside of the IMPALA *representation* network is different from the second LSTM used inside the model *dynamics* function, described later. In the Atari experiments, the network takes as the input one RGB frame. Stacking more frames would help as evidenced in Figure B.9.

Q-network and model architecture. The original IMPALA agent was not learning a Q-function. Because we train a MuZero-like model, we can estimate the

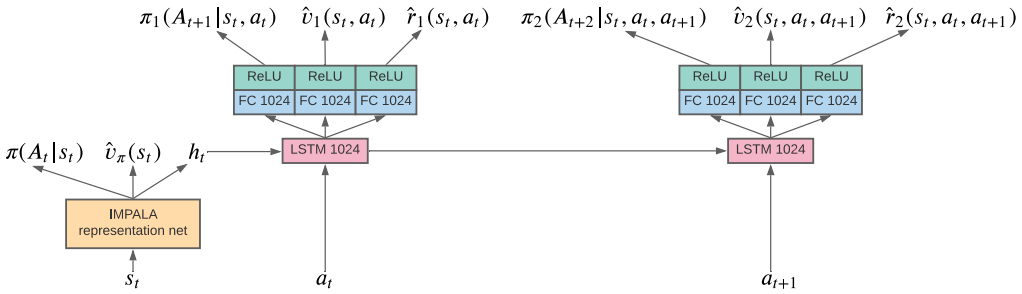


Figure B.2: The model architecture when using the IMPALA-based representation network. The $\hat{r}_1(s_t,a_t)$ predicts the reward $\mathbb{E}[R_{t+1}|s_t,a_t]$. The $\hat{v}_1(s_t,a_t)$ predicts the value $\mathbb{E}[v_\pi(S_{t+1})|s_t,a_t]$. In general, $\hat{r}_k(s_t,a_{<t+k})$ predicts the reward $\mathbb{E}[R_{t+k}|s_t,a_{<t+k}]$. And $\hat{v}_k(s_t,a_{<t+k})$ predicts the value $\mathbb{E}[v_\pi(S_{t+k})|s_t,a_{<t+k}]$.

Q-values by:

$$\hat{q}(s,a) = \hat{r}_1(s,a) + \gamma \hat{v}_1(s,a), \tag{B.31}$$

where $\hat{r}_1(s,a)$ and $\hat{v}_1(s,a)$ are the reward model and the value model, respectively. The reward model and the value model are based on MuZero *dynamics* and *prediction* functions (Schrittwieser et al., 2020). We use a very small dynamics function, consisting of a single LSTM layer with 1024 hidden units, conditioned on the selected action (Figure B.2).

The decomposition of $\hat{q}(s,a)$ to a reward model and a value model is not crucial. The Muesli agent obtained a similar score with a model of the $q_\pi(s,a)$ action-values (Figure B.6).

Value model and reward model losses. Like in MuZero (Schrittwieser et al., 2020), the value model and the reward model are trained by categorical losses. The target for the value model is the multi-step return estimate provided by Retrace (Munos et al., 2016). Inside of the Retrace, we use $\hat{q}_{\pi_{\text{prior}}}(s,a)$ action-values provided by the target network.

Optimizer. We use the Adam optimizer (Kingma and Ba, 2014) with the decoupled weight decay by Loshchilov and Hutter (2017). The learning rate is linearly decayed to reach zero at the end of the training. We do not clip the norm of the gra-

dient. Instead, we clip the parameter updates to $[-1,1]$, before multiplying them with the learning rate. In Adam's notation, the update rule is:

$$\theta_t = \theta_{t-1} + \alpha \,\text{clip}(\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}, -1, 1), \tag{B.32}$$

where $\hat{m}_t$ and $\hat{v}_t$ are the estimated moments, not value functions.

Replay. As observed by Schmitt et al. (2020), the LASER agent benefited from mixing replay data with on-policy data in each batch. Like LASER, we also use uniform replay and mix replay data with on-policy data. To obtain results comparable with other methods, we do not use LASER's *shared* experience replay and hence compare to the LASER version that did not share experience either.

Evaluation. On Atari, the human-normalized score is computed at 200M environment frames (including skipped frames). The episode returns are collected from the last 200 training episodes that *finished before* the 200M environment frames. This is the same evaluation as used by MuZero. The replayed frames are not counted in the 200M frame limit. For example, if replayed frames form 95% of each batch, the agent is trained for 20 times more steps than an agent with no replay.

B.6.2 Muesli policy update

The Muesli policy loss usage is summarized in Algorithm 6.

Prior policy. We use a *target* network to approximate $v_{\pi_{\text{prior}}}$, $q_{\pi_{\text{prior}}}$ and π_{prior}. Like the target network in DQN (Mnih et al., 2015), the target network contains older network parameters. We use an exponential moving average to continuously update the parameters of the target network.

In general, the π_{prior} can be represented by a mixture of multiple policies. When forming π_{CMPO}, we represented π_{prior} by the target network policy mixed with a small proportion of the uniform policy (0.3%) and the behavior policy (3%). Mixing with these policies was not a significant improvement to the results (Figure B.10).

B.6.3 Hyperparameters

On Atari, the experiments used the Arcade Learning Environment (Bellemare et al., 2013) with sticky actions. The environment parameters are listed in Table B.2.

The hyperparameters shared by all policy updates are listed in Table B.3. When comparing the clipped and unclipped advantages in Figure 4.4, we estimated the $\mathrm{KL}(\pi_{\mathrm{CMPO}}, \pi)$ with exact KL. The unclipped advantages would have too large variance without the exact KL.

The hyperparameters for the large-scale Atari experiments are in Table B.4, hyperparameters for 9x9 Go self-play are in Table B.5, and hyperparameters for continuous control on MuJoCo are in Table B.6. On Go, the discount $\gamma = -1$ allows us to train by self-play on the two-player perfect-information zero-sum game with alternate moves without modifying the reinforcement learning algorithms.

Table B.2: Atari parameters. In general, we follow the recommendations by Machado et al. (2018).

PARAMETER	VALUE
Random modes and difficulties	No
Sticky action probability ς	0.25
Start no-ops	0
Life information	Not allowed
Action set	18 actions
Max episode length	30 minutes (108,000 frames)
Observation size	96×96
Action repetitions	4
Max-pool over last N action repeat frames	4
Total environment frames, including skipped frames	200M

B.6.4 Policy losses

We will explain the other compared policy losses here. When comparing the different policy losses, we always used the same network architecture and the same reward model and value model training. The advantages were always normalized.

The hyperparameters for all policy losses are listed in Table B.7. We tuned the hyperparameters for all policy losses on 10 Atari games (`alien`, `beam_rider`, `breakout`, `gravitar`, `hero`, `ms_pacman`, `phoenix`, `robotank`, `seaquest`

Table B.3: Hyperparameters shared by all experiments.

HYPERPARAMETER	VALUE
Batch size	96 sequences
Sequence length	30 frames
Model unroll length K	5
Replay proportion in a batch	75%
Replay buffer capacity	6,000,000 frames
Initial learning rate	3×10^{-4}
Final learning rate	0
AdamW weight decay	0
Discount	0.995
Target network update rate α_{target}	0.1
Value loss weight	0.25
Reward loss weight	1.0
Retrace $\mathbb{E}_{A\sim\pi}[\hat{q}_{\pi_{\text{prior}}}(s,A)]$ estimator	16 samples
$\text{KL}(\pi_{\text{CMPO}}, \pi)$ estimator	16 samples
Variance moving average decay β_{var}	0.99
Variance offset ε_{var}	10^{-12}

Table B.4: Modified hyperparameters for large-scale Atari experiments. The network architecture, discount and replay proportion are based on MuZero Reanalyze.

HYPERPARAMETER	VALUE
Network architecture	MuZero net with 16 ResNet blocks
Stacked frames	16
Batch size	768 sequences
Replay proportion in a batch	95%
Replay buffer capacity	28,800,000 frames
AdamW weight decay	10^{-4}
Discount	0.997
Retrace λ	0.95
$\text{KL}(\pi_{\text{CMPO}}, \pi)$ estimator	exact KL

and `time_pilot`). For each hyperparameter, we tried multiples of 3 (e.g., 0.1, 0.3, 1.0, 3.0). For the PPO clipping threshold, we explored 0.2, 0.25, 0.3, 0.5, 0.8.

Policy gradients (PG). The simplest tested policy loss uses policy gradients with the entropy regularizer, as in Mnih et al. (2016). The loss is defined by

$$L_{\text{PG}}(\pi, s) = -\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{\text{adv}}(s,A)\right] - \lambda_{\text{H}}\text{H}[\pi(\cdot|s)]. \tag{B.33}$$

Table B.5: Modified hyperparameters for 9x9 Go self-play experiments.

HYPERPARAMETER	VALUE
Network architecture	MuZero net with 6 ResNet blocks
Batch size	192 sequences
Sequence length	49 frames
Replay proportion in a batch	0%
Initial learning rate	2×10^{-4}
Target network update rate α_{target}	0.01
Discount	-1 (self-play)
Multi-step return estimator	V-trace
V-trace λ	0.99

Table B.6: Modified hyperparameters for MuJoCo experiments.

HYPERPARAMETER	VALUE
Replay proportion in a batch	95.8%

Policy gradients with the TRPO penalty. The next policy loss uses $\mathrm{KL}(\pi_b(\cdot|s), \pi(\cdot|s))$ inside the regularizer. The π_b is the behavior policy. This policy loss is known to work as well as PPO (Cobbe et al., 2020).

$$L_{\text{PG+TRPOpenalty}}(\pi, s) = -\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{\text{adv}}(s,A)\right] - \lambda_{\text{H}}\text{H}[\pi(\cdot|s)] + \lambda_{\text{TRPO}}\,\mathrm{KL}(\pi_b(\cdot|s), \pi(\cdot|s)). \tag{B.34}$$

Proximal Policy Optimization (PPO). PPO (Schulman et al., 2017) is usually used with multiple policy updates on the same batch of data. In our setup, we use a replay buffer instead. PPO then required a larger clipping threshold ε_{PPO}. In our setup, the policy gradient with the TRPO penalty is a stronger baseline.

Table B.7: Hyperparameters for the different policy losses.

HYPERPARAMETER	PG	TRPO penalty	PPO	MPO	Muesli
Total policy loss weight	3.0	3.0	3.0	3.0	3.0
Entropy bonus weight	0.003	0.0003	0.0003	0	0
TRPO penalty weight	0	0.01	0	0	0
PPO clipping ε_{PPO}	-	-	0.5	-	-
MPO $\text{KL}(\pi_{\text{MPO}}, \pi)$ constraint	-	-	-	0.01	-
CMPO loss weight	0	0	0	-	1.0
CMPO clipping threshold c	-	-	-	-	1.0

$$L_{\text{PPO}}(\pi, s) = -\mathbb{E}_{A\sim\pi_b(\cdot|s)}\Big[\min\Big(\frac{\pi(A|s)}{\pi_b(A|s)}\,\hat{\text{adv}}(s,A), \\ \text{clip}\Big(\frac{\pi(A|s)}{\pi_b(A|s)}, 1-\varepsilon_{\text{PPO}}, 1+\varepsilon_{\text{PPO}}\Big)\,\hat{\text{adv}}(s,A)\Big)\Big] \\ -\lambda_{\text{H}}\text{H}[\pi(\cdot|s)]. \tag{B.35}$$

Maximum a Posteriori Policy Optimization (MPO). We use a simple variant of MPO (Abdolmaleki et al., 2018) that is not specialized to Gaussian policies. Also, we use $\pi_{\text{MPO}}(\cdot|s_{t+k})$ as the target for the policy model.

$$L_{\text{MPO}}(\pi, s_t) = \text{KL}(\pi_{\text{MPO}}(\cdot|s_t), \pi(\cdot|s_t)) + \frac{1}{K}\sum_{k=1}^{K}\text{KL}(\pi_{\text{MPO}}(\cdot|s_{t+k}), \pi_k(\cdot|s_t, a_{<t+k})) \\ s.t.\ \mathbb{E}_{S\sim d_{\pi_b}}\left[\text{KL}(\pi_{\text{MPO}}(\cdot|S), \pi(\cdot|S))\right] < \varepsilon_{\text{MPO}}. \tag{B.36}$$

Direct MPO. Direct MPO uses the MPO regularizer $\lambda\,\text{KL}(\pi, \pi_{\text{prior}})$ as a penalty.

$$L_{\text{DirectMPO}}(\pi, s) = -\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{\text{adv}}(s,A)\right] + \lambda\,\text{KL}(\pi(\cdot|s), \pi_{\text{prior}}(\cdot|s)). \tag{B.37}$$

B.6.5 Go experimental details

The Go environment was configured using OpenSpiel (Lanctot et al., 2019). Games were scored with the Tromp–Taylor rules with a komi of 7.5. Observations consisted of the last 2 board positions, presented with respect to the player in three 9x9 planes each (player's stones, opponent's stones, and empty intersections), in addition to a plane indicating the player's color. The agents were evaluated against GnuGo v3.8 at level 10 (Bump et al., 2005) and Pachi v11.99 (Baudiš and Gailly, 2011) with 10,000 simulations, 16 threads, and no pondering. Both were configured with the Chinese ruleset. Figure B.3 shows the results versus GnuGo.

B.7 Additional experiments

Table B.8 lists the median and mean human-normalized score across 57 Atari games. The table also lists the differences in the number of stacked frames, the amount of replay and the probability of a sticky action. The environment with a non-zero probability of a sticky action is more challenging by being stochastic (Machado et al., 2018).

In Figure B.7, we compare the different ways to act and explore during training. Muesli (in blue) acts by sampling actions from the policy network. Acting proportionally to π_{CMPO} was not significantly different (in green). Acting based on the Q-values only was substantially worse (in red). This is consistent with our example from Figure 4.2 where acting with Q-values would be worse.

Table B.8: Median and mean human-normalized score across 57 Atari games, after 200M environment frames. The agents differ in the network size, the number of stacked frames, the amount of replay, the probability of a sticky action, and agent training. The ± indicates the standard error across 2 random seeds. While DreamerV2 was not evaluated on `defender` and `surround`, DreamerV2 median score remains valid on 57 games if we assume a high DreamerV2 score on `defender`.

AGENT	MEDIAN	MEAN	STACK	REPLAY	STICKY ACTION
DQN (Mnih et al., 2015)	79%	-	4	87.5%	0.0
IMPALA (Espeholt et al., 2018)	192%	958%	4	0%	0.0
IQN (Dabney et al., 2018a)	218%	-	4	87.5%	0.0
Rainbow (Hessel et al., 2018)	231%	-	4	87.5%	0.0
Meta-gradient$\{\gamma, \lambda\}$ (Xu et al., 2018)	287%	-	4	0%	0.0
LASER (Schmitt et al., 2020)	431%	-	4	87.5%	0.0
DreamerV2 (Hafner et al., 2020)	164%	-	1	-	0.25
Muesli with IMPALA architecture	562 ±3%	1,981 ±66%	4	75%	0.25
Muesli with MuZero arch, replay=80%	755 ±27%	2,253 ±120%	16	80%	0.25
Muesli with MuZero arch, replay=95%	**1,041** ±40%	2,524 ±104%	16	95%	0.25
MuZero Reanalyse (Schrittwieser et al., 2021)	**1,047** ±40%	2,971 ±115%	16	95%	0.25

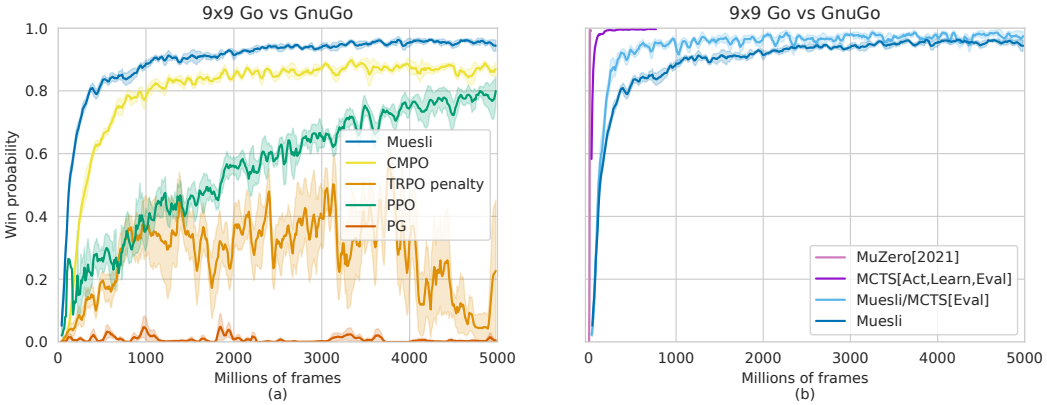


Figure B.3: Win probability on 9x9 Go when training from scratch, by self-play, for 5B frames. Evaluating 3 seeds against GnuGo (level 10). **(a)** Muesli and other search-free baselines. **(b)** MuZero MCTS with 150 simulations and Muesli with and without the use of MCTS at the evaluation time only.

Algorithm 6 Agent with Muesli policy loss

Initialization:

Initialize the estimate of the variance of the advantage estimator:

$\text{var} := 0$

$\beta_{\text{product}} := 1.0$

Initialize π_{prior} parameters with the π parameters:

$\theta_{\pi_{\text{prior}}} := \theta_\pi$

Data collection on an actor:

For each step:

Observe state s_t and select action $a_t \sim \pi_{\text{prior}}(\cdot|s_t)$.

Execute a_t in the environment.

Append $s_t, a_t, r_{t+1}, \gamma_{t+1}$ to the replay buffer.

Append $s_t, a_t, r_{t+1}, \gamma_{t+1}$ to the online queue.

Training on a learner:

For each minibatch:

Form a minibatch B of sequences from the online queue and the replay buffer.

Use Retrace to estimate each return $G^v(s,a)$, bootstrapping from $\hat{q}_{\pi_{\text{prior}}}$.

Estimate the variance of the advantage estimator:

$\text{var} := \beta_{\text{var}}\text{var} + (1-\beta_{\text{var}})\frac{1}{|B|}\sum_{(s,a)\in B}(G^v(s,a) - \hat{v}_{\pi_{\text{prior}}}(s))^2$

Compute the bias-corrected variance estimate in Adam's style:

$\beta_{\text{product}} := \beta_{\text{product}}\beta_{\text{var}}$

$\widehat{\text{var}} := \frac{\text{var}}{1-\beta_{\text{product}}}$

Prepare the normalized advantages:

$\hat{\text{adv}}(s,a) = \frac{\hat{q}_{\pi_{\text{prior}}}(s,a) - \hat{v}_{\pi_{\text{prior}}}(s)}{\sqrt{\widehat{\text{var}} + \varepsilon_{\text{var}}}}$

Compute the total loss:

$L_{\text{total}} = ($

$\quad L_{\text{PG+CMPO}}(\pi, s)$ // Regularized policy optim., Eq. 4.6.

$\quad + L_m(\pi, s)$ // Policy model loss, Eq. 4.10.

$\quad + L_v(\hat{v}_\pi, s) + L_r(\hat{r}_\pi, s))$ // MuZero value and reward losses.

Use L_{total} to update θ_π by one step of gradient descent.

Use a moving average of π parameters as π_{prior} parameters:

$\theta_{\pi_{\text{prior}}} := (1-\alpha_{\text{target}})\theta_{\pi_{\text{prior}}} + \alpha_{\text{target}}\theta_\pi$

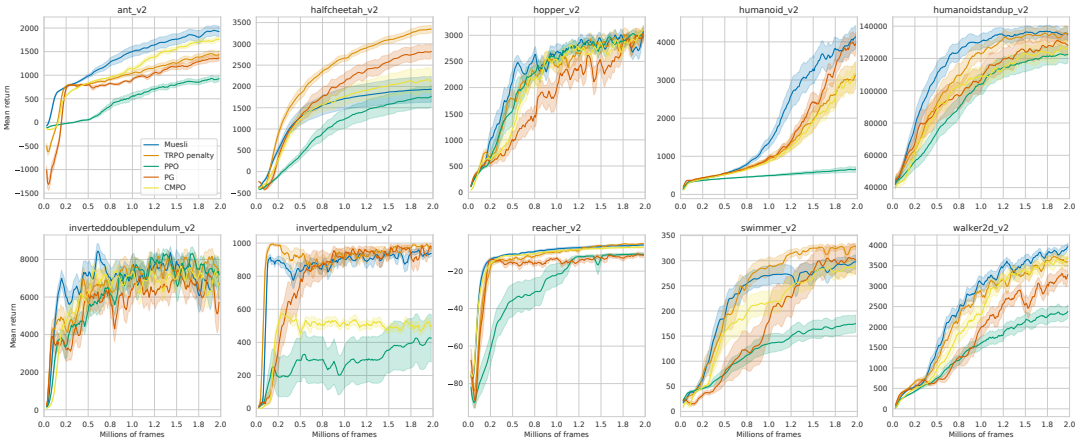


Figure B.4: Mean episode return on MuJoCo environments from OpenAI Gym. The shaded area indicates the standard error across 10 random seeds.

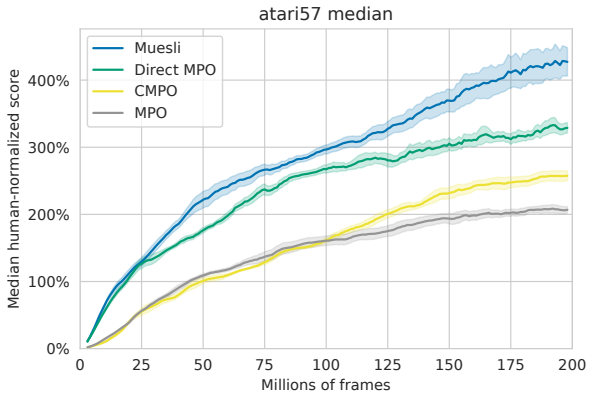


Figure B.5: Median score of across 57 Atari games for different MPO variants. CMPO is MPO with clipped advantages and no constrained optimization.

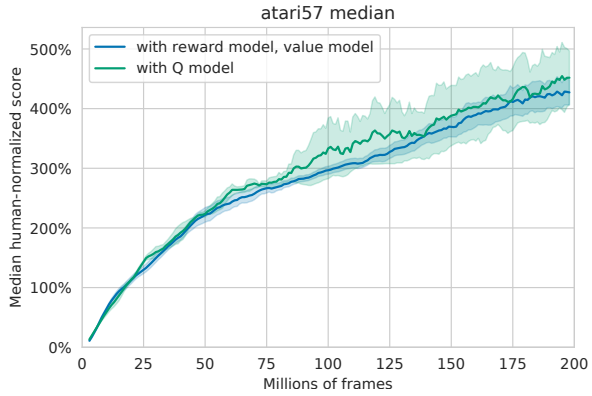


Figure B.6: Median score of Muesli across 57 Atari games when modeling the reward and value or when modeling the $q_\pi(s,a)$ directly.

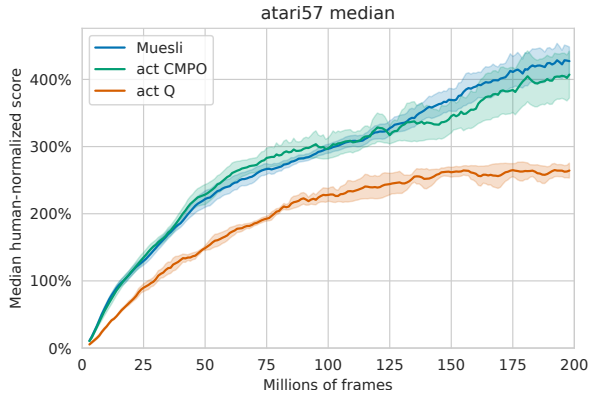


Figure B.7: Median score across 57 Atari games for different ways to act and explore. Acting with π_{CMPO} was not significantly different. Acting with $\text{softmax}(\hat{q}/\textit{temperature})$ was worse.

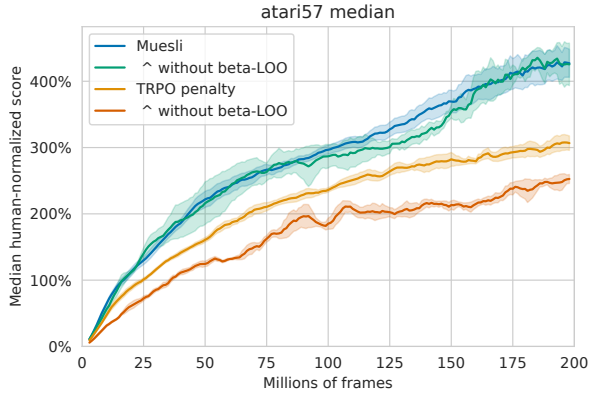


Figure B.8: Median score across 57 Atari games when using or not using β-LOO action-dependent baselines.

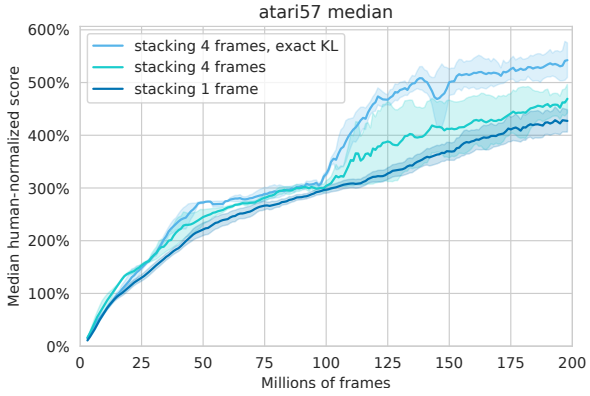


Figure B.9: Median score across 57 Atari games for different numbers of stacked frames.

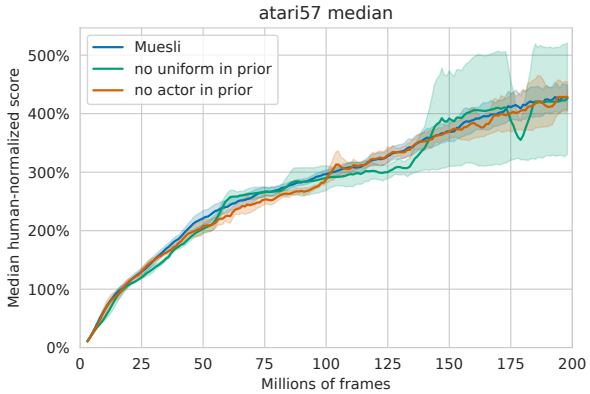


Figure B.10: Median score across 57 Atari games for different π_{prior} compositions.

Table B.9: The mean score from the last 200 episodes at 200M frames on 57 Atari games. The ± indicates the standard error across 2 random seeds.

GAME	Random	Human	MuZero		Muesli	
alien	228	7128	135541	±65349	**139409**	±12178
amidar	6	1720	1061	±136	**21653**	±2019
assault	222	742	29697	±3595	**36963**	±533
asterix	210	8503	**918628**	±56222	316210	±48368
asteroids	719	47389	**509953**	±33541	484609	±5047
atlantis	12850	29028	1136009	±1466	**1363427**	±81093
bank_heist	14	753	**14176**	±13044	1213	±0
battle_zone	2360	37188	320641	±141924	**414107**	±13422
beam_rider	364	16927	**319684**	±13394	288870	±137
berzerk	124	2630	19523	±16817	**44478**	±36140
bowling	23	161	156	±25	**191**	±37
boxing	0	12	**100**	±0	99	±1
breakout	2	30	778	±20	**791**	±10
centipede	2091	12017	862737	±11564	**869751**	±16547
chopper_command	811	7388	**494578**	±488588	101289	±24339
crazy_climber	10780	35829	**176172**	±17630	175322	±3408
defender	2874	18689	544320	±12881	**629482**	±39646
demon_attack	152	1971	**143846**	±8	129544	±11792
double_dunk	-19	-16	**24**	±0	-3	±2
enduro	0	861	**2363**	±2	2362	±1
fishing_derby	-92	-39	**69**	±5	51	±0
freeway	0	30	**34**	±0	33	±0
frostbite	65	4335	**410173**	±35403	301694	±275298
gopher	258	2412	**121342**	±1540	104441	±424
gravitar	173	3351	10926	±2919	**11660**	±481
hero	1027	30826	**37249**	±15	37161	±114
ice_hockey	-11	1	**40**	±2	25	±13
jamesbond	29	303	**32107**	±3480	19319	±3673
kangaroo	52	3035	13928	±90	**14096**	±421
krull	1598	2666	**50137**	±22433	34221	±1385
kung_fu_master	258	22736	**148533**	±31806	134689	±9557
montezuma_revenge	0	**4753**	1450	±1050	2359	±309
ms_pacman	307	6952	**79319**	±8659	65278	±1589
name_this_game	2292	8049	**108133**	±6935	105043	±732
phoenix	761	7243	748424	±67304	**805305**	±26719
pitfall	-229	**6464**	0	±0	0	±0
pong	-21	15	**21**	±0	20	±1
private_eye	25	**69571**	7600	±7500	10323	±4735
qbert	164	13455	85926	±8980	**157353**	±6593
riverraid	1338	17118	**172266**	±592	47323	±1079
road_runner	12	7845	**554956**	±23859	327025	±45241
robotank	2	12	**85**	±15	59	±2
seaquest	68	42055	501236	±498423	**815970**	±128885
skiing	-17098	**-4337**	-30000	±0	-18407	±1171
solaris	1236	**12327**	4401	±732	3031	±491
space_invaders	148	1669	31265	±27619	**59602**	±2759
star_gunner	664	10250	158608	±4060	**214383**	±23087
surround	-10	7	**10**	±0	9	±0
tennis	-24	-8	-0	±0	**12**	±12
time_pilot	3568	5229	**413988**	±10023	359105	±21396
tutankham	11	168	**318**	±30	252	±47
up_n_down	533	11693	**606602**	±28296	549190	±70789
venture	0	1188	866	±866	**2104**	±291
video_pinball	0	17668	**921563**	±56020	685436	±155718
wizard_of_wor	564	4757	**103463**	±3366	93291	±5
yars_revenge	3093	54577	187731	±32107	**557818**	±1895
zaxxon	32	9173	**106935**	±45495	65325	±395

Appendix C

Appendix to Policy Improvement by Planning with Gumbel

Content

C.1 Proof for planning with Gumbel

We will prove that Algorithm 4 generates A_{n+1} such that $\mathbb{E}[q(A_{n+1})] \geq \mathbb{E}_{A\sim\pi}[q(A)]$. For the right-hand side, the Gumbel-Max trick tells us that $\mathbb{E}_{A\sim\pi}[q(A)]$ is equal to $\mathbb{E}_{(g\in\mathbb{R}^k)\sim\text{Gumbel}(0)}[q(\arg\max_a(g(a)+\text{logits}(a))]$. First, we will show that we can replace the $\arg\max_a$ with $\arg\max_{a\in\mathcal{A}_{\text{topn}}}$. Remember that $\mathcal{A}_{\text{topn}}$ is defined as $\mathcal{A}_{\text{topn}} = \texttt{argtop}(g+\text{logits}, n)$ and that we use the same Gumbel vector g in the `argtop` and $\arg\max$. The set $\mathcal{A}_{\text{topn}}$ then includes the action with the highest $g(a)+\text{logits}(a)$, and we can replace the $\arg\max_a$ with $\arg\max_{a\in\mathcal{A}_{\text{topn}}}$.

After these rewrites, we have to prove that

$$\mathbb{E}[q(A_{n+1})] \geq \mathbb{E}_{(g\in\mathbb{R}^k)\sim\text{Gumbel}(0)}[q(\arg\max_{a\in\mathcal{A}_{\text{topn}}}(g(a)+\text{logits}(a))]. \tag{C.1}$$

On the left-hand side, $\mathbb{E}[q(A_{n+1})]$ is equal to

$$\mathbb{E}_{(g\in\mathbb{R}^k)\sim\text{Gumbel}(0)}[q(\arg\max_{a\in\mathcal{A}_{\text{topn}}}(g(a)+\text{logits}(a)+\sigma(q(a))))]. \tag{C.2}$$

We can finish the proof by proving that for any vector $g\in\mathbb{R}^k$ we have

$$q(\arg\max_{a\in\mathcal{A}_{\text{topn}}}(g(a)+\text{logits}(a)+\sigma(q(a)))) \geq q(\arg\max_{a\in\mathcal{A}_{\text{topn}}}(g(a)+\text{logits}(a)). \tag{C.3}$$

This is true because σ is a monotonically increasing transformation.

C.2 Proof for completed Q-values

We will prove that $\pi'_{\text{completed}} = \text{softmax}(\text{logits}+\sigma(\text{completedQ}))$ produces a policy improvement. We will start by showing that $\pi'_{\text{completed}}$ is produced by a specific instance of Algorithm 7. We will then prove that any instance of Algorithm 7 produces a policy improvement.

Algorithm 7 Policy improvement when having v_π

Require: π, v_π.
Require: $q(a)$ for each visited action. The visited actions can be from any distribution.
Initialize π' with π.
For the visited actions:
If $q(a) > v_\pi$, increase the $\pi'(a)$ logit.
If $q(a) < v_\pi$, decrease the $\pi'(a)$ logit.
return π'

C.2.1 Specific instance

Algorithm 7 is more general than the usage of the completed Q-values. Specifically, Algorithm 7 would produce $\pi'_{\text{completed}}$ if updating the logits by $\sigma(\text{completedQ}(a)) - \sigma(v_\pi)$. This update increases the logit if $q(a) > v_\pi$. This update decreases the logit if $q(a) < v_\pi$. And the update does not modify the logits of the unvisited actions. The resulting $\text{softmax}(\text{logits}+\sigma(\text{completedQ}) - \sigma(v_\pi))$ is equal to $\text{softmax}(\text{logits}+\sigma(\text{completedQ}))$ because the constant offset $\sigma(v_\pi)$ does not change the softmax output.

C.2.2 Policy improvement proof for any instance

We will now prove that π' from Algorithm 7 satisfies

$$\sum_a \pi'(a)q(a) \geq \sum_a \pi(a)q(a). \tag{C.4}$$

Notice that $\pi'(a)$ for any unvisited action a will be $c_z\pi(a)$, with a normalization constant $c_z > 0$.

For one visited action: Let's denote the visited (aka expanded) action by a_{ex}. First, if $\pi(a_{\text{ex}}) = 1$ then $v_\pi = q(a_{\text{ex}})$ and the policy will remain unchanged.

Let's now consider the case with $\pi(a_{\text{ex}}) < 1$. The v_π can be rewritten as

$$v_\pi = \pi(a_{\text{ex}})q(a_{\text{ex}}) + (1 - \pi(a_{\text{ex}})) \sum_{a \neq a_{\text{ex}}} \frac{\pi(a)q(a)}{\sum_{b \neq a_{\text{ex}}} \pi(b)}. \tag{C.5}$$

Let's denote the weighted sum by q_{miss}:

$$q_{\text{miss}} = \sum_{a \neq a_{\text{ex}}} \frac{\pi(a)q(a)}{\sum_{b \neq a_{\text{ex}}} \pi(b)}. \tag{C.6}$$

We notice that the q_{miss} does not change if scaling π by a constant $c_z > 0$.

We will now rewrite the left-hand side of Inequality C.4 to use q_{miss}:

$$\begin{aligned}
\sum_a \pi'(a)q(a) = & && \text{(C.7)}\\
&= \pi'(a_{\text{ex}})q(a_{\text{ex}}) + (1 - \pi'(a_{\text{ex}})) \sum_{a \neq a_{\text{ex}}} \frac{c_z\pi(a)q(a)}{\sum_{b \neq a_{\text{ex}}} c_z\pi(b)} && \text{(C.8)}\\
&= \pi'(a_{\text{ex}})q(a_{\text{ex}}) + (1 - \pi'(a_{\text{ex}}))q_{\text{miss}} && \text{(C.9)}\\
&= \pi'(a_{\text{ex}})(q(a_{\text{ex}}) - q_{\text{miss}}) + q_{\text{miss}}. && \text{(C.10)}
\end{aligned}$$

The right-hand side of Inequality C.4 can be also rewritten:

$$\begin{aligned}
v_\pi &= \pi(a_{\text{ex}})q(a_{\text{ex}}) + (1 - \pi(a_{\text{ex}}))q_{\text{miss}} && \text{(C.11)}\\
&= \pi(a_{\text{ex}})(q(a_{\text{ex}}) - q_{\text{miss}}) + q_{\text{miss}}. && \text{(C.12)}
\end{aligned}$$

With these rewrites, Inequality C.4 becomes

$$\pi'(a_{\text{ex}})(q(a_{\text{ex}}) - q_{\text{miss}}) \geq \pi(a_{\text{ex}})(q(a_{\text{ex}}) - q_{\text{miss}}). \tag{C.13}$$

The $q(a_{\text{ex}}) - q_{\text{miss}}$ can be negative, zero or positive.
If $q(a_{\text{ex}}) = q_{\text{miss}}$, the inequality is satisfied.
If $q(a_{\text{ex}}) > q_{\text{miss}}$, we want $\pi'(a_{\text{ex}}) \geq \pi(a_{\text{ex}})$.
If $q(a_{\text{ex}}) < q_{\text{miss}}$, we want $\pi'(a_{\text{ex}}) \leq \pi(a_{\text{ex}})$.

Because we do not know q_{miss}, we cannot use it in an algorithm. We will instead show that $q(a_{\text{ex}}) > q_{\text{miss}}$ is equivalent to $q(a_{\text{ex}}) > v_\pi$ when $\pi(a_{\text{ex}}) < 1$:

$$q(a_{\text{ex}}) > v_\pi \tag{C.14}$$

$$q(a_{\text{ex}}) > \pi(a_{\text{ex}})q(a_{\text{ex}}) + (1 - \pi(a_{\text{ex}}))q_{\text{miss}} \tag{C.15}$$

$$(1 - \pi(a_{\text{ex}}))q(a_{\text{ex}}) > (1 - \pi(a_{\text{ex}}))q_{\text{miss}} \tag{C.16}$$

$$q(a_{\text{ex}}) > q_{\text{miss}}. \tag{C.17}$$

Thus we directly arrived at Algorithm 7.

For multiple visited actions: We will focus on one of the visited actions. If the logits of the other visited actions are unmodified, the algorithm is equivalent to using only one visited action. If the logits of the other visited actions are modified by Algorithm 7, they can further help to improve the policy.

C.3 Mixed value approximation

We will construct an approximation of v_π. The exact v_π is defined by $v_\pi = \sum_a \pi(a)q(a)$. We have an approximate $\hat{v}_\pi$ from a value network, we know π, and we have $q(a)$ for the visited actions. With these inputs, we approximate v_π by a consistent estimator:

$$v_{\text{mix}} = \frac{1}{1 + \sum_b N(b)} \left(\hat{v}_\pi + \frac{\sum_b N(b)}{\sum_{b \in \{b:N(b)>0\}} \pi(b)} \sum_{a \in \{a:N(a)>0\}} \pi(a)q(a) \right). \tag{C.18}$$

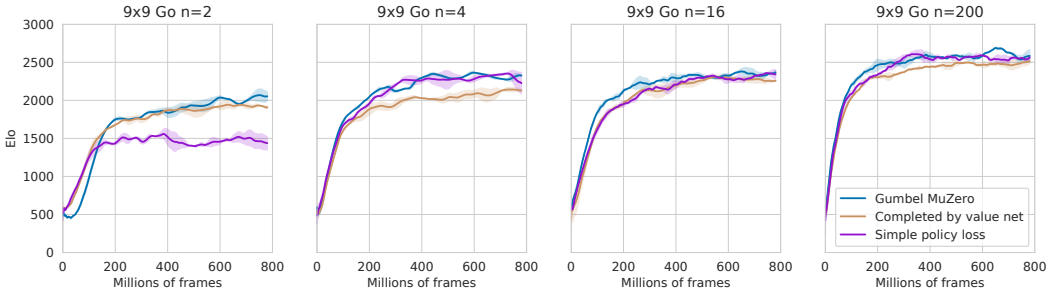


Figure C.1: Detailed policy loss ablations. Gumbel MuZero uses the policy loss with Q-values completed by the v_{mix} value estimator from Appendix C.3. That works better than Q-values completed by the raw value network $\hat{v}_\pi$.

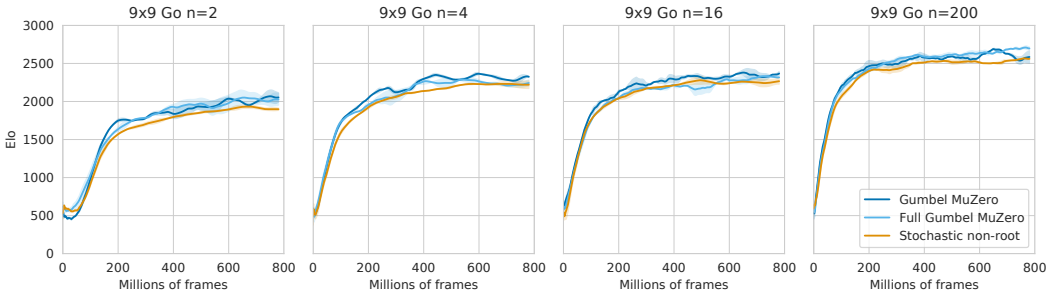


Figure C.2: A comparison of different action selections at the non-root nodes. Gumbel MuZero uses the unmodified (deterministic) MuZero action selection at non-root nodes. Full Gumbel MuZero uses the deterministic action selection from Equation 5.14, which we compare to stochastic sampling from π' at non-root nodes.

The estimator interpolates $\hat{v}_\pi$ and the weighted average of the available Q-values. This is an unsophisticated estimator, with results in Figure C.1. You are welcome to explore other possibilities.

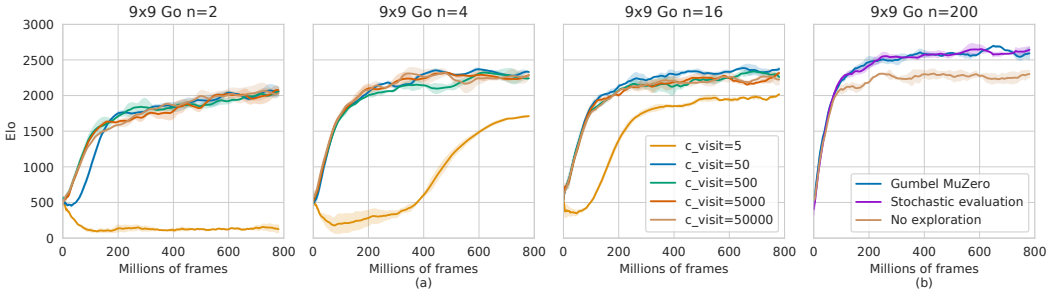


Figure C.3: Additional Gumbel MuZero ablations on 9x9 Go. **(a)** Sensitivity to Q-value scaling by c_{visit}. **(b)** On the perfect-information game, Gumbel MuZero used zero Gumbel noise at evaluation. Although, evaluation with stochastic Gumbel noise is not worse. During training, MuZero and Gumbel MuZero benefit from explorative acting proportional to the visit counts.

C.4 Derivation of the deterministic action selection

We will derive Equation 5.14 from Equation 5.13:

$$\arg\min_a \sum_b \left(\pi'(b) - \frac{N(b)+\mathbb{I}\{a=b\}}{1+\sum_c N(c)}\right)^2 \tag{C.19}$$

$$= \arg\min_a \sum_b \left(\left(\pi'(b) - \frac{N(b)}{1+\sum_c N(c)}\right) - \frac{\mathbb{I}\{a=b\}}{1+\sum_c N(c)}\right)^2 \tag{C.20}$$

$$= \arg\min_a \sum_b -2\left(\pi'(b) - \frac{N(b)}{1+\sum_c N(c)}\right)\frac{\mathbb{I}\{a=b\}}{1+\sum_c N(c)} \tag{C.21}$$

$$= \arg\min_a -\sum_b \left(\pi'(b) - \frac{N(b)}{1+\sum_c N(c)}\right)\mathbb{I}\{a=b\} \tag{C.22}$$

$$= \arg\max_a \left(\pi'(a) - \frac{N(a)}{1+\sum_c N(c)}\right). \tag{C.23}$$

The simplification was possible because additive terms independent of a do not affect $\arg\min_a$. While widely applicable, the deterministic action selection provides only a small benefit on 9x9 Go (Figure C.2).

C.5 Experimental details

In general, we use hyperparameters consistent with the newest MuZero experiments (Schrittwieser et al., 2021). MuZero's pseudocode is available thanks to Schrittwieser et al. (2020). Gumbel MuZero does not need to set the Dirichlet noise hyperparameters because Gumbel MuZero does not use Dirichlet noise.

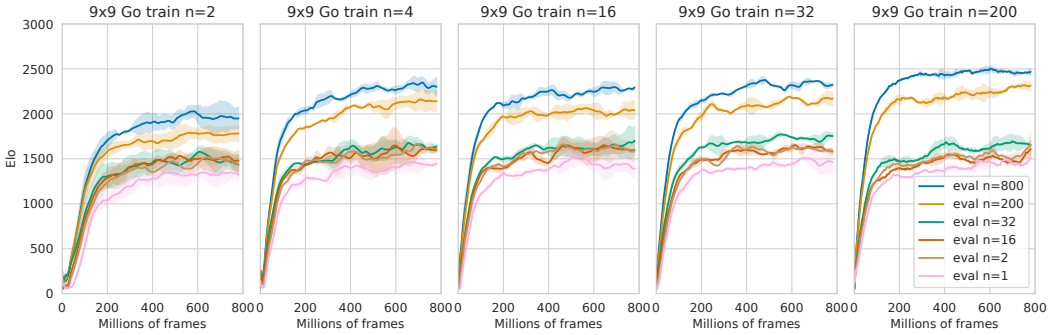


Figure C.4: Gumbel MuZero Elo on 9x9 Go, evaluated with different numbers of simulations. The evaluation with $n = 1$ simulation acts with the most-probable action from the policy network.

In a tree search, the Q-values $\hat{q}(a)$ are provided by the visited child nodes. We do not modify the structure of AlphaZero's search tree. Inside the tree search, the Q-values are normalized to be from the $[0,1]$ interval. We use the normalized Q-values also in Full Gumbel MuZero, but the algorithm does not require Q-values from a specific interval. In all Go and chess experiments, Gumbel MuZero scales the Q-values by $c_{\text{visit}} = 50$ and $c_{\text{scale}} = 1.0$. On the perfect-information game of Go, Gumbel MuZero is not very sensitive to the scale of the Q-values. Any $c_{\text{visit}} \geq 50$ produced similar results (Figure C.3a).

In each phase of Sequential Halving, we use at least one new visit. For example, in the first phase, we update $\hat{q}(a)$ by $\max\left(1, \left\lfloor \frac{n}{\lceil \log_2(m) \rceil m} \right\rfloor\right)$ visits. This allows us to experiment with incomplete or no Sequential Halving. When Sequential Halving runs out of the budget of n simulations, we stop the search. The agent then selects as A_{n+1} the action with the highest $g(a) + \text{logits}(a) + \sigma(\hat{q}(a))$ from the set of the most-visited actions. Fabiano and Cazenave (2021) provide a different way to deal with the rounding in Sequential Halving.

During training, MuZero acts with explorative actions in the first 30 moves of each self-play game. MuZero samples the explorative actions proportionally to the visit counts, like AlphaGo Zero (Silver et al., 2017a). Gumel MuZero benefits from the same exploration (Figure C.3b).

Figure C.4 shows the importance of the number of simulations at evaluation

time. For example, in the first subplot, a network is trained with $n = 2$ simulations and the same network is evaluated with 800, 200, 32, 16, 2, and 1 simulations.

To run the experiments, we used Google Cloud Tensor Processing Units v3 (TPUs). On 9x9 Go, MuZero is not limited by lack of data if using three times more TPUs for self-play than for training. By using a smaller number of simulations, we can substantially reduce the number of TPUs needed for self-play. Table C.1 lists the obtained speedups if not being limited by the TPUs for training.

We apologize for running the expensive experiments with only 2 seeds. We make no claims about confidence intervals. Sometimes a single data point is enough to prove that something is possible (e.g., going to the Moon).

Table C.1: The speedup from a smaller number of simulations on 9x9 Go.

	Training step speedup
MuZero $n = 200$	1.0
Full Gumbel MuZero $n = 200$	1.0
Gumbel MuZero $n = 200$	1.0
Gumbel MuZero $n = 32$	5.9
Gumbel MuZero $n = 16$	11.3
Gumbel MuZero $n = 8$	16.2
Gumbel MuZero $n = 4$	24.3

C.5.1 Network Architecture

We used a small 6-layer network on 9x9 Go, and a bigger 32-layer network in the large-scale 19x19 Go experiments. The networks used 256 hidden planes, 128 bottleneck planes and a broadcasting block in every 8th layer.

Bibliography

Abbas, Z., Sokota, S., Talvitie, E., and White, M. (2020). Selective dyna-style planning under limited model capacity. In *International Conference on Machine Learning*, pages 1–10. PMLR.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018). Maximum a Posteriori Policy Optimisation. In *International Conference on Learning Representations*.

Agarwal, A., Henaff, M., Kakade, S., and Sun, W. (2020a). PC-PG: Policy cover directed exploration for provable policy gradient learning. *Advances in Neural Information Processing Systems*, 33:13399–13412.

Agarwal, A., Jiang, N., Kakade, S. M., and Sun, W. (2020b). *Reinforcement Learning: Theory and Algorithms*. `https://rltheorybook.github.io`.

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2019). On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift. *arXiv e-prints*, page arXiv:1908.00261.

Amos, B., Dinh, L., Cabi, S., Rothörl, T., Muldal, A., Erez, T., Tassa, Y., de Freitas, N., and Denil, M. (2018). Learning awareness models. In *International Conference on Learning Representations*.

Anthony, T., Nishihara, R., Moritz, P., Salimans, T., and Schulman, J. (2019). Policy Gradient Search: Online Planning and Expert Iteration without Search Trees. *arXiv e-prints*, page arXiv:1904.03646.

Anthony, T., Tian, Z., and Barber, D. (2017). Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30.

Antonoglou, I., Schrittwieser, J., Ozair, S., Hubert, T. K., and Silver, D. (2022). Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*.

Åström, K. (1965). Optimal Control of Markov Processes with Incomplete State Information I. *Journal of Mathematical Analysis and Applications*, 10:174–205.

Audibert, J.-Y., Bubeck, S., and Munos, R. (2010). Best arm identification in multi-armed bandits. In *COLT*, pages 41–53. Citeseer.

Bareinboim, E., Forney, A., and Pearl, J. (2015). Bandits with unobserved confounders: A causal approach. In *Advances in Neural Information Processing Systems*, pages 1342–1350.

Baudiš, P. and Gailly, J.-l. (2011). Pachi: State of the art open source Go program. In *Advances in computer games*, pages 24–38. Springer.

Bellemare, M. G., Danihelka, I., Dabney, W., Mohamed, S., Lakshminarayanan, B., Hoyer, S., and Munos, R. (2017). The Cramer distance as a solution to biased Wasserstein gradients. *arXiv e-prints*, page arXiv:1705.10743.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An evaluation platform for general agents. *JAIR*.

Bertsekas, D. (2019). *Reinforcement and Optimal Control*. Athena Scientific.

Bertsekas, D. (2021). *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific.

Bertsekas, D. (2022). *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*. Athena Scientific `http://web.mit.edu/dimitrib/www/RLbook.html`.

Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D. X., Chickering, D. M., Portugaly, E., Ray, D., Simard, P., and Snelson, E. (2013). Counterfactual reasoning and learning systems: The example of computational advertising. *Journal of Machine Learning Research*, 14(65):3207–3260.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs. `http://github.com/google/jax`.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3):33–57.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv e-prints*, page arXiv:1606.01540.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Bubeck, S., Munos, R., and Stoltz, G. (2011). Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science*, 412(19):1832–1852.

Bump et al. (2005). GnuGo. `http://www.gnu.org/software/gnugo/gnugo.html`.

Byravan, A., Springenberg, J. T., Abdolmaleki, A., Hafner, R., Neunert, M., Lampe, T., Siegel, N., Heess, N., and Riedmiller, M. (2020). Imagined value gradients: Model-based policy optimization with tranferable latent dynamics models. In *Conference on Robot Learning*, pages 566–589. PMLR.

Cazenave, T. (2014). Sequential halving applied to trees. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(1):102–105.

Chiappa, S., Racanière, S., Wierstra, D., and Mohamed, S. (2017). Recurrent environment simulators. In *International Conference on Learning Representation*.

Cobbe, K., Hilton, J., Klimov, O., and Schulman, J. (2020). Phasic Policy Gradient. *arXiv e-prints*, page arXiv:2009.04416.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018a). Implicit Quantile Networks for Distributional Reinforcement Learning. In *International Conference on Machine Learning*, pages 1096–1105.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2018b). Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Dam, T. Q., D'Eramo, C., Peters, J., and Pajarinen, J. (2021). Convex regularization in Monte-Carlo tree search. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2365–2375. PMLR.

Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. (2022). Policy improvement by planning with Gumbel. In *International Conference on Learning Representations*.

Danihelka, I., Lakshminarayanan, B., Uria, B., Wierstra, D., and Dayan, P. (2017). Comparison of maximum likelihood and GAN-based training of Real NVPs. *arXiv e-prints*, page arXiv:1705.05263.

Degris, T. and Modayil, J. (2012). Scaling-up Knowledge for a Cognizant Robot. In *AAAI Spring Symposium: Designing Intelligent Robots*.

Degris, T., Pilarski, P. M., and Sutton, R. S. (2012). Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, pages 2177–2182.

Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*.

Diuk, C., Cohen, A., and Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *International Conference on Machine Learning*, pages 240–247.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. (2018). Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control. *arXiv e-prints*, page arXiv:1812.00568.

Elo, A. E. (1978). *The Rating of Chessplayers, Past and Present*. Arco Pub., New York.

Eslami, S. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., et al. (2018). Neural scene representation and rendering. *Science*, 360(6394):1204–1210.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*.

Fabiano, N. and Cazenave, T. (2021). Sequential halving using scores. In *Reinforcement Learning in Games at AAAI*.

Farquhar, G., Rocktäschel, T., Igl, M., and Whiteson, S. (2018). TreeQN and ATreeC: Differentiable tree-structured models for deep reinforcement learning. In *International Conference on Learning Representations*, volume 6. ICLR.

Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *IEEE International Conference on Robotics and Automation*, pages 2786–2793.

Forney, A., Pearl, J., and Bareinboim, E. (2017). Counterfactual data-fusion for online reinforcement learners. In *International Conference on Machine Learning*, pages 1156–1164.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591.

Gelly, S. and Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175:1856–1875.

Gregor, K., Jimenez Rezende, D., Besse, F., Wu, Y., Merzic, H., and van den Oord, A. (2019). Shaping belief states with generative environment models for RL. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., and Munos, R. (2020). Monte-Carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3769–3778. PMLR.

Grimm, C., Barreto, A., Singh, S., and Silver, D. (2020). The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33.

Gruslys, A., Dabney, W., Azar, M. G., Piot, B., Bellemare, M., and Munos, R. (2018). The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning. In *International Conference on Learning Representations*.

Guez, A., Mirza, M., Gregor, K., Kabra, R., Racaniere, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., et al. (2019). An investigation of model-free planning. In *International Conference on Machine Learning*, pages 2464–2473.

Guez, A., Viola, F., Weber, T., Buesing, L., Kapturowski, S., Precup, D., Silver, D.,

and Heess, N. (2020). Value-driven hindsight modelling. In *Advances in Neural Information Processing Systems*.

Gumbel, E. J. (1954). *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office.

Guo, Z. D., Gheshlaghi Azar, M., Piot, B., Pires, B. A., and Munos, R. (2018). Neural Predictive Belief Representations. *arXiv e-prints*, page arXiv:1811.06407.

Guo, Z. D., Pires, B. A., Piot, B., Grill, J.-B., Altché, F., Munos, R., and Azar, M. G. (2020). Bootstrap latent-predictive representations for multitask reinforcement learning. In *International Conference on Machine Learning*, pages 3875–3886. PMLR.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, volume 31, pages 2450–2462. Curran Associates, Inc.

Ha, D. and Schmidhuber, J. (2018). World Models. *arXiv e-prints*, page arXiv:1803.10122.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. *arXiv e-prints*, page arXiv:1812.05905.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Mastering Atari with Discrete World Models. *arXiv e-prints*, page arXiv:2010.02193.

Hamrick, J. B. (2019). Analogues of mental simulation and imagination in deep learning. *Current Opinion in Behavioral Sciences*, 29:8–16.

Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Pfaff, T., Weber, T., Buesing, L., and Battaglia, P. W. (2020). Combining Q-learning and search with amortized value estimates. In *International Conference on Learning Representations*.

Hamrick, J. B., Friesen, A. L., Behbahani, F., Guez, A., Viola, F., Witherspoon, S., Anthony, T., Buesing, L. H., Veličković, P., and Weber, T. (2021). On the role of planning in model-based deep reinforcement learning. In *International Conference on Learning Representations*.

Hardt, M. and Recht, B. (2021). *Patterns, predictions, and actions: A story about machine learning*. `https://mlstory.org`.

Hay, N. and Russell, S. J. (2011). Metareasoning for Monte Carlo tree search. Technical report, EECS Department, University of California, Berkeley.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. In *Computer Vision – ECCV 2016*, pages 630–645.

Hessel, M., Danihelka, I., Viola, F., Guez, A., Schmitt, S., Sifre, L., Weber, T., Silver, D., and van Hasselt, H. (2021). Muesli: Combining improvements in policy optimization. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4214–4226. PMLR.

Hessel, M., Kroiss, M., Clark, A., Kemaev, I., Quan, J., Keck, T., Viola, F., and van Hasselt, H. (2021). Podracer architectures for scalable Reinforcement Learning. *arXiv e-prints*, page arXiv:2104.06272.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *AAAI Conference on Artificial Intelligence*.

Hessel, M., van Hasselt, H., Modayil, J., and Silver, D. (2019). On Inductive Biases in Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1907.02908.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv e-prints*, page arXiv:1503.02531.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S., and Silver, D. (2021). Learning and planning in complex action spaces. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4476–4486. PMLR.

Igl, M., Zintgraf, L., Le, T. A., Wood, F., and Whiteson, S. (2018). Deep Variational Reinforcement Learning for POMDPs. *arXiv e-prints*, page arXiv:1806.02426.

Jaakkola, T., Singh, S., and Jordan, M. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems*, volume 7. MIT Press.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509.

Kaelbling, L. P. and Lozano-Pérez, T. (2010). Hierarchical task and motion planning in the now. In *AAAI Conference on Bridging the Gap Between Task and Motion Planning*, AAAIWS'10-01, page 33–42. AAAI Press.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. (2019). Model-Based Reinforcement Learning for Atari. *arXiv e-prints*, page arXiv:1903.00374.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pages 267–274.

Kakade, S. M. (2001). A natural policy gradient. *Advances in Neural Information Processing Systems*, 14:1531–1538.

Kalchbrenner, N., van den Oord, A., Simonyan, K., Danihelka, I., Vinyals, O., Graves, A., and Kavukcuoglu, K. (2017). Video pixel networks. In *International Conference on Machine Learning*, pages 1771–1779.

Karnin, Z., Koren, T., and Somekh, O. (2013). Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1238–1246. PMLR.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pages 282–293.

Kool, W., van Hoof, H., and Welling, M. (2019). Stochastic beams and where to find them: The Gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR.

Kool, W., van Hoof, H., and Welling, M. (2020). Estimating gradients for discrete random variables by sampling without replacement. In *International Conference on Learning Representations*.

Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., Hennes, D., Morrill, D., Muller, P., Ewalds, T., Faulkner, R., Kramár, J., De Vylder, B., Saeta, B., Bradbury, J., Ding, D., Borgeaud, S., Lai, M., Schrittwieser, J., Anthony, T., Hughes, E., Danihelka, I., and Ryan-Davis, J. (2019). OpenSpiel: A Framework for Reinforcement Learning in Games. *arXiv e-prints*, page arXiv:1908.09453.

Lee, S. and Bareinboim, E. (2018). Structural causal bandits: Where to intervene? In *Advances in Neural Information Processing Systems*, pages 2568–2578.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv e-prints*, page arXiv:2005.01643.

Li, X., Li, L., Gao, J., He, X., Chen, J., Deng, L., and He, J. (2015). Recurrent Reinforcement Learning: A Hybrid Approach. *arXiv e-prints*, page arXiv:1509.03044.

Lin, L. and Mitchell, T. (1992). Memory approaches to reinforcement learning in non-Markovian domains. Technical report, Carnegie Mellon University.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321.

Linscott, G., Pascutto, G.-C., Lyashuk, A., and Huizinga, F. (2018). Leela Chess Zero. `http://lczero.org/`.

Loshchilov, I. and Hutter, F. (2017). Decoupled Weight Decay Regularization. *arXiv e-prints*, page arXiv:1711.05101.

Lu, C., Schölkopf, B., and Hernández-Lobato, J. M. (2018). Deconfounding Reinforcement Learning in Observational Settings. *arXiv e-prints*, page arXiv:1812.10576.

Luce, R. D. (1959). Individual choice behavior: A theoretical analysis.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.

Maddison, C., Mnih, A., and Teh, Y. (2017). The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*.

Michie, D. (1966). Game-playing and game-learning automata. In *Advances in programming and non-numerical computation*, pages 183–200. Elsevier.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

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. (2015). Human-level control through deep reinforcement learning. *Nature*.

Moerland, T. M., Broekens, J., and Jonker, C. M. (2020). Model-based Reinforcement Learning: A Survey. *arXiv e-prints*, page arXiv:2006.16712.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062.

Munro, P. (1987). A dual back-propagation scheme for scalar reward learning. In *9th Annual Conference of the Cognitive Science Society*, pages 165–176.

Nasu, Y. (2018). Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi. *The 28th World Computer Shogi Championship Appeal Document*.

Nauman, M. and Den Hengst, F. (2020). Low-Variance Policy Gradient Estimation with World Models. *arXiv e-prints*, page arXiv:2010.15622.

Nguyen, D. and Widrow, B. (1990). The truck backer-upper: An example of self-learning in neural networks. In *Advanced neural computers*, pages 11–19. Elsevier.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems*, pages 2845–2853. Curran Associates, Inc.

Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128.

Ortega, P. A., Kunesch, M., Delétang, G., Genewein, T., Grau-Moya, J., Veness, J., Buchli, J., Degrave, J., Piot, B., Perolat, J., Everitt, T., Tallec, C., Parisotto, E., Erez, T., Chen, Y., Reed, S., Hutter, M., de Freitas, N., and Legg, S. (2021). Shaking the foundations: Delusions in sequence models for interaction and control. *arXiv e-prints*, page arXiv:2110.10819.

Papamakarios, G., Nalisnick, E., Jimenez Rezende, D., Mohamed, S., and Lakshminarayanan, B. (2019). Normalizing Flows for Probabilistic Modeling and Inference. *arXiv e-prints*, page arXiv:1912.02762.

Pascanu, R., Li, Y., Vinyals, O., Heess, N., Buesing, L., Racanière, S., Reichert, D., Weber, T., Wierstra, D., and Battaglia, P. (2017). Learning model-based planning from scratch. *arXiv e-prints*, page arXiv:1707.06170.

Pearl, J., Glymour, M., and Jewell, N. P. (2016). *Causal inference in statistics: A primer*. John Wiley & Sons.

Pearl, J. and Mackenzie, D. (2018). *The Book of Why: The new science of cause and effect*. Basic Books.

Pepels, T., Cazenave, T., Winands, M. H., and Lanctot, M. (2014). Minimizing simple and cumulative regret in Monte-Carlo tree search. In *Workshop on Computer Games*, pages 1–15. Springer.

Pohlen, T., Piot, B., Hester, T., Gheshlaghi Azar, M., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Večerík, M., Hessel, M., Munos, R., and Pietquin, O. (2018). Observe and Look Further: Achieving Consistent Performance on Atari. *arXiv e-prints*, page arXiv:1805.11593.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Rezende, D. J., Danihelka, I., Papamakarios, G., Ke, N. R., Jiang, R., Weber, T., Gregor, K., Merzic, H., Viola, F., Wang, J., Mitrovic, J., Besse, F., Antonoglou, I., and Buesing, L. (2020). Causally Correct Partial Models for Reinforcement Learning. *arXiv e-prints*, page arXiv:2002.02836.

Richalet, J., Rault, A., Testud, J. L., and Papon, J. (1978). Paper: Model predictive heuristic control. *Automatica*, 14(5):413–428.

Riedmiller, M. (2005). Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, ECML'05, page 317–328, Berlin, Heidelberg. Springer-Verlag.

Rosin, C. D. (2011). Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv e-prints*, page arXiv:1703.03864.

Schmidhuber, J. (1990). An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *In Proc. IEEE/INNS International Joint Conference on Neural Networks*, pages 253–258. IEEE Press.

Schmidhuber, J. (1991). Curious model-building control systems. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1458–1463.

Schmitt, S., Hessel, M., and Simonyan, K. (2020). Off-Policy Actor-Critic with Shared Experience Replay. In *International Conference on Machine Learning*, volume 119, pages 8545–8554, Virtual. PMLR.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schrittwieser, J., Hubert, T. K., Mandhane, A., Barekatain, M., Antonoglou, I., and Silver, D. (2021). Online and offline reinforcement learning by planning with a learned model. In *Advances in Neural Information Processing Systems*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust Region Policy Optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv e-prints*, page arXiv:1707.06347.

Silver, D., Goyal, A., Danihelka, I., Hessel, M., and van Hasselt, H. (2022). Learning by directional gradient descent. In *International Conference on Learning Representations*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement

learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395. JMLR.org.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017a). Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Silver, D., Sutton, R. S., and Müller, M. (2008). Sample-based learning and search with permanent and transient memories. In *International Conference on Machine Learning*, pages 968–975.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. (2017b). The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3191–3199. PMLR.

Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, volume 23, pages 2164–2172. Curran Associates, Inc.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In *Machine Learning Proceedings 1994*, pages 284–292. Elsevier.

Springenberg, J. T., Heess, N., Mankowitz, D., Merel, J., Byravan, A., Abdolmaleki, A., Kay, J., Degrave, J., Schrittwieser, J., Tassa, Y., Buchli, J., Belov, D., and Riedmiller, M. (2020). Local Search for Policy Iteration in Continuous Control. *arXiv e-prints*, page arXiv:2010.05545.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. (2018). Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4732–4741. PMLR.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*.

Sutton, R. S. (1990). Integrated architectures for learning, planning and reacting based on dynamic programming. In *Machine Learning: Proceedings of the Seventh International Workshop*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 761–768.

Talvitie, E. (2014). Model regularization for stable sample rollouts. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 780–789.

Talvitie, E. (2017). Self-correcting models for model-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Talvitie, E. (2018). Learning the reward function for a misspecified model. In *International Conference on Machine Learning*, pages 4838–4847. PMLR.

Tamar, A., WU, Y., Thomas, G., Levine, S., and Abbeel, P. (2016). Value iteration networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Tasfi, N. and Capretz, M. (2018). Dynamic Planning Networks. *arXiv e-prints*, page arXiv:1812.11240.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68.

Tolpin, D. and Shimony, S. (2012). MCTS based on simple regret. In *AAAI Conference on Artificial Intelligence*, volume 26.

Tomar, M., Shani, L., Efroni, Y., and Ghavamzadeh, M. (2020). Mirror Descent Policy Optimization. *arXiv e-prints*, page arXiv:2005.09814.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017). Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

van Hasselt, H., Quan, J., Hessel, M., Xu, Z., Borsa, D., and Barreto, A. (2019). General non-linear Bellman equations. *arXiv e-prints*, page arXiv:1907.03687.

van Hasselt, H. and Sutton, R. S. (2015). Learning to Predict Independent of Span. *arXiv e-prints*, page arXiv:1508.04582.

van Hasselt, H. and Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279. IEEE.

van Hasselt, H. P., Guez, A., Guez, A., Hessel, M., Mnih, V., and Silver, D. (2016). Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

van Hasselt, H. P., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, volume 32, pages 14322–14333. Curran Associates, Inc.

Vieillard, N., Kozuno, T., Scherrer, B., Pietquin, O., Munos, R., and Geist, M. (2020). Leverage the average: An analysis of KL regularization in RL. In *Advances in Neural Information Processing Systems*.

Vieira, T. (2014). Gumbel-max trick and weighted reservoir sampling. `http://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weighted-reservoir-sampling/`.

Vieira, T. (2017). Estimating means in a finite universe. `https://timvieira.github.io/blog/post/2017/07/03/estimating-means-in-a-finite-universe/`.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample Efficient Actor-Critic with Experience Replay. *arXiv e-prints*, page arXiv:1611.01224.

Wang, Z., Novikov, A., Zolna, K., Merel, J. S., Springenberg, J. T., Reed, S. E., Shahriari, B., Siegel, N., Gulcehre, C., Heess, N., et al. (2020). Critic regularized regression. *Advances in Neural Information Processing Systems*, 33.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England.

Werbos, P. J. (1987). Learning how the world works: Specifications for predictive networks in robots and brains. In *IEEE International Conference on Systems, Man and Cybernetics, N.Y.*

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., and Schmidhuber, J. (2014). Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980.

Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

Williams, R. and Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268.

Xiao, C., Huang, R., Mei, J., Schuurmans, D., and Müller, M. (2019). Maximum entropy Monte-Carlo planning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Xie, C., Patil, S., Moldovan, T., Levine, S., and Abbeel, P. (2016). Model-based reinforcement learning with parametrized physical models and optimism-driven exploration. In *IEEE International Conference on Robotics and Automation*, pages 504–511.

Xu, Z., van Hasselt, H., and Silver, D. (2018). Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2402–2413.

Yellott, J. I. (1977). The relationship between Luce's choice axiom, Thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, 15(2):109–144.

Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H. P., Silver, D., and Singh, S. (2020). A self-tuning actor-critic algorithm. In *Advances in Neural Information Processing Systems*, volume 33, pages 20913–20924. Curran Associates, Inc.

Zhang, J. and Bareinboim, E. (2017). Transfer learning in multi-armed bandits: A causal approach. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1340–1346.

Zhang, S. and Yao, H. (2019). ACE: An actor ensemble algorithm for continuous control with tree search. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 5789–5796.

Zhu, S., Ng, I., and Chen, Z. (2019). Causal Discovery with Reinforcement Learning. *arXiv e-prints*, page arXiv:1906.04477.