# Model-based Reinforcement Learning Methods for Developing Intelligent Assistants

by

Katya Kudashkina

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Doctor of Philosophy
in
Engineering

Guelph, Ontario, Canada

# ABSTRACT

MODEL-BASED REINFORCEMENT LEARNING METHODS
FOR DEVELOPING INTELLIGENT ASSISTANTS

Katya Kudashkina
University of Guelph, 2022

Advisor:
Dr. Julie Vale

Intelligent assistants could benefit humans and the environment in many ways: they could help make predictions about the future, enhance productivity, optimize work, and make improvements not only in daily tasks but on a global level. Intelligent assistants are a challenge to develop. In particular, conversational intelligent assistants are still a subject of artificial intelligence (AI) research.

The challenge of conversational AI requires a domain that is tightly scoped so that the intelligent assistant can learn and understand it. In this thesis, I propose a *voice document-editing* domain, argue that the proposed domain is particularly promising for conducting research in conversational AI, and present its primary advantages. Further, I present suitable methods for developing intelligent assistants that are designed to learn from interaction without explicit instruction. I show that *model-based reinforcement learning* is a particularly appropriate class of methods to pursue.

Next, I provide a sample-efficient model-based reinforcement learning method—soft-planner policy optimization. In this method, we introduce a novel soft-planner policy

and present a new update in which the soft-planner policy is used to fine-tune a model-free policy and value function. Using a specific deletion task within the proposed domain, I demonstrate that the soft-planner policy optimization method allows the agent to efficiently learn. I compare the method to the current state-of-the-art implementations for such systems: model-free actor-critic methods and supervised methods.

Finally, within model-based reinforcement learning methods, I focus on methods that use expectation models and linear value functions. I show that planning with an expectation model must update a state-value function, not an action-value function as previously suggested in the literature. I then demonstrate three methods in which actions can be selected when planning with state-value functions and present general model-based reinforcement learning algorithms for each. I present the practical application of these methods on our proposed voice document-editing domain.

Altogether, this thesis combines deep learning, natural language processing, and reinforcement learning in the context of intelligent assistants, in particular, conversational AI. We focus on model-based reinforcement learning methods that solve control problems and combine the settings of online learning, function approximation, and stochastic environments.

**Keywords:** expectation models, function approximation, conversational AI, deep learning, intelligent assistants, linear value functions, machine learning, model-based reinforcement learning, neural networks, natural language processing, online learning, planning, stochastic.

# DEDICATION

To my mother, Svetlana.

# ACKNOWLEDGEMENTS

This thesis would not have been possible without the support of a large number of fantastic individuals and institutions. First and foremost I would like to thank my family—my mother Svetlana, my dearest sister Natasha, and my father Sergei. They never give up on me no matter what. Thank you for always believing in me, protecting, supporting, and encouraging me to be who I am. I would like to thank my fiancé Maxim Souraev for his endless patience, support, love, and care.

I am deeply indebted to all of you in my PhD committee for this journey and could never thank you all enough for lifting me up on the hardest days. The major ideas of this thesis started out in collaboration with Rich Sutton. I would like to thank Rich for his guidance, criticism, mentorship, and inspiration—Rich, you taught me to look deeper, to search for answers endlessly, and to never stop polishing my writing. I would like to thank Julie Vale—Julie, you are the star and I look up to you on many dimensions. I would like to thank Jakob Foerster—Jakob, I would not have done this without your help and support. Julie, Rich, and Jakob—I would like to thank you for carefully examining this thesis and for your constructive feedback. I would like to thank Peter Wittek—Peter, even though you now rest in Mount Trishul in the Himalayas, you gave this work a head start and a kick of encouragement. I would like to thank Graham Taylor for advising me in the earlier years of my PhD, until our ways parted.

I am thankful to Mike Bowling, Yi Wan, Abhishek Naik, Patrick Pilarski, Valiappa Chockalingam, and Jamie Kiros for the fun collaboration and for valuable advice. A special thank you goes to Ajay Agrawal, Martha Steenstrup, Marc Bellemare, Beatrix Dart, and Hado van Hasselt for their insights, advice, and support along the way.

I am also thankful to my friends and colleagues at the RLAI at the University of Alberta: Eric Graves, Roshan Shariff, Joseph Modayil, Khurram Javed, Matthew

Schlegel, Marlos C. Machado, Anna Koop, John D. Martin, Zach Holland, Matt Taylor, Tian Tian, Sina Ghiassian, Kenny Young, Shibhansh Dohare, Kory Mathewson, Alex Kearney, Martha White, Adam White, Juan Fernando Hernandez Garcia, Kris De Asis, Andy Patterson. I am grateful to my friends and colleagues at the MLRG at the University of Guelph: Michal Lisicki, Nikhil Sapru, Carolyn Augusta, Thor Jonson, Angus Galloway, Adam Balint, Devinder Kumar, Brendan Duke, Alaa El-Nouby, Vithursan Thangarasa, Eu Wern Teh, Boris Knyazev, Kristina Kuperschmidt, Eric Taylor, Ethan Jackson, Mostafa Nategholeslam, Elahe Ghalebi. I am indebted to the Arrell Food Institute at the University of Guelph for the Arrell scholarship. I am thankful to the Canadian Institute for Advanced Research, and the Vector Institute, Compute Canada and the Alberta Machine Intelligence Institute. I am grateful to my close and/or long-term friends for mostly bearing with me: Russ Asai, Eleni Triantafillou, Anna Golbeva, Irina Kuznetsova, Olga Kadysh, Ken Constable, Natalia Shevchenko, Timothy Jordan, Jon French, Irina Baratov, Jane Podbelskaya , and many others.

For any errors or naïveté I alone am responsible.

# STATEMENT OF CONTRIBUTIONS

This thesis is a culmination of years of hard thinking and is based on the following papers and pre-prints. It connects intelligent assistants, conversational AI, deep learning, natural language processing, and reinforcement learning.

In Chapter 3, I provide my position on what is necessary to advance research on intelligent assistants, and in particular, conversational AI assistants. In this chapter I present my first contribution and propose a voice editing domain and argue for methods suitable for solving it. The ideas described in this chapter are drawn from my work that started while visiting the University of Alberta as a Visiting Scholar. The work was done in collaboration with Dr. Patrick Pilarski and Dr. Richard S. Sutton. At the time of the submission of the thesis this work is under review with Springer Nature Computer Science Journal and is available online: Kudashkina K., Pilarski P., and Sutton R. S., "Document-editing Assistants and Model-based Reinforcement Learning as a Path to Conversational AI". ArXiv: 2008.12095.

In Chapter 4, I present my second contribution that is a model-based reinforcement learning method—*soft-planner policy optimization*. This work was done in collaboration with Valliappa Chockalingam, Dr. Peter Wittek, Dr. Michael Bowling, and Dr. Graham W. Taylor. This work is available online: Kudashkina K., Chockalingam V, Taylor G., Bowling M., "Sample-Efficient Model-based Actor-Critic for an Interactive Dialogue Task". ArXiv: 2004.13657.

In Chapter 5, I explore model-based reinforcement learning methods for planning with expectation models for control. The first part of this work was done during my

second visit to the University of Alberta as a Visiting Scholar, working in collaboration with Yi Wan, Abhishek Naik, and Dr. Richard S. Sutton. This pre-print of the work is available online: Kudashkina K., Wan Y., Naik A., Sutton R. S., "Planning with Expectation Models for Control". ArXiv: 2104.08543. Here, I present my main contribution in which I prove that planning with appealing expectation models cannot be done with action-value functions. This brings up the question that was side-stepped before of how planning is going to effect action selection. I address the question as my fourth contribution by exploring planning with state-value function and considering three strategies for action selection.

In Chapter 6, I explore the realization of the voice editing domain further. This chapter presents the practical application of this action selection strategies discussed in Chapter 5 on our proposed voice document-editing domain.

Overall, this thesis contributes model-based reinforcement learning methods applicable for building intelligent assistants.

# TABLE OF CONTENTS

# LIST OF FIGURES

# NOTATION SUMMARY

Capital letters are used for random variables. Lower case letters are used for the instantiations of random variables—the values of random variables—and for scalar functions. For example, the state, action, and reward at time step $t$ are denoted $S_t$, $A_t$, and $R_t$, while their possible values might be denoted $s$, $a$, and $r$. Approximate value functions are deterministic functions of random parameters and are thus also in lower case (e.g., $\hat{v}(\mathbf{s}, \mathbf{w}) \approx v_\pi(s)$).

| | |
|---|---|
| $\doteq$ | equality relationship that is true by definition |
| $\approx$ | approximately equal |
| $Pr\{X = x\}$ | probability that a random variable $X$ takes the value $x$ |
| $X \sim p$ | $X$ selected from a distribution $p(x) \doteq Pr\{X = x\}$ |
| $\mathbb{E}[X]$ | expectation of a random variable $X$, i.e., $\mathbb{E} \doteq \sum_x p(x)x$ |
| $\arg\max_a f(a)$ | a value of $a$ at which $f(a)$ takes its maximal value |
| $\top$ | transpose operator |
| $\ln x$ | natural logarithm of $x$ |
| $e^x$ | the base of natural logarithm of $x$ |
| $\mathbb{R}$ | set of real numbers |
| $\leftarrow$ | assignment, used to avoid confusion with definitions |
| $(a, b]$ | a real interval between $a$ and $b$ including $b$ but not including $a$ |
| $\epsilon$ | probability of taking a random action in an $\epsilon$-greedy policy |
| $\tau$ | trajectory |
| $\mathrm{H}_t$ | history up to a time step $t$ including past actions, observations, and rewards |

| | |
|---|---|
| $\alpha, \beta$ | step-size parameters, also referred to as learning rates |
| $t$ | discrete time step |
| $\zeta$ | stochasticity of the environment |
| $\gamma$ | discount parameter |
| $T, T(t)$ | final time step of an episode $A_t$ action at time t |
| $\mathcal{A}(s)$ | set of all actions available in state $s$ |
| $s, s'$ | environment states |
| $S_t$ | environment state at time $t$ |
| $\mathbf{s}, \mathbf{s}'$ | feature-vectors of an agent state |
| $\mathbf{s}_t$ | feature-vector of an agent state at time $t$ |
| $u$ | state-update function |
| $\mathcal{S}$ | set of all non-terminal states |
| $R, R_t$ | reward at time $t$ |
| $\pi$ | policy (decision-making) rule |
| $\pi(\boldsymbol{\theta})$ | policy $\pi$ corresponding to parameters $\boldsymbol{\theta}$ |
| $\pi(a \mid \mathbf{s})$ | probability of taking action $a$ in state $s$ under *stochastic* policy $\pi$; a probability distribution over all $a \in \mathcal{A}(s)$ for each $\mathbf{s} \in \mathcal{S}$ |
| $\pi^{\text{planner}}$ | planner policy |
| $v_\pi(\mathbf{s})$ | value of state s under policy $\pi$ (expected return) |
| $v_\pi$ or $v_*$ | value functions |
| $p(s', r \mid s, a)$ | joint probability of transition to next state $s'$ with reward $r$ given the current state $s$ and action $a$ |
| $p(s' \mid s, a)$ | probability of transition to state $s'$, from state $s$ and action $a$ |
| $d$ | dimensionality—e.g., the number of components of $\mathbf{w}$ |
| $d'$ | alternate dimensionality—the number of components of $\boldsymbol{\theta}$ |
| $\mathbf{w}, \mathbf{w}_t$ | $d$-vector of weights underlying an approximate policy |
| $w_i$ | $i$-th component of a learnable weight vector |
| $\boldsymbol{\theta}, \boldsymbol{\theta}_t$ | $d$-vector of weights underlying an approximate value function |

| | |
|---|---|
| $\mathbf{w}^q, \mathbf{w}_t^q$ | $d$-vector of weights underlying an approximate action-value function |
| $\hat{v}(\mathbf{s}, \mathbf{w})$ | approximate value of state $\mathbf{s}$ given weight vector $\mathbf{w}$ |
| $\hat{q}(\mathbf{s}, a, \mathbf{w})$ | approximate value of a state–action pair $\mathbf{s}, a$ given weight vector $\mathbf{w}$ |
| $\nabla\hat{v}(\mathbf{s}, \mathbf{w})$ | column vector of partial derivatives of $\hat{v}(s, \mathbf{w})$ with respect to $\mathbf{w}$ |
| $\nabla\hat{q}(\mathbf{s}, a, \mathbf{w})$ | column vector of partial derivatives of $\hat{q}(s, a, \mathbf{w})$ with respect to $\mathbf{w}$ |
| $\alpha^{\boldsymbol{\theta}}$ | step-size corresponding to learnable weights $\boldsymbol{\theta}$ |
| $\alpha^{\mathbf{w}}$ | step-size corresponding to learnable weights $\mathbf{w}$ |
| $\psi_t$ | speech transcription at a time step $t$ |
| $w_t$ | word sequence |
| $O_t$ | observation |
| $I_t$ | user's intent |
| $m_t$ | number of steps within an interaction |
| $b_t$ | number of words that were not corrupted but deleted by the agent incorrectly |
| $H$ | entropy |
| $J(\boldsymbol{\theta})$ | performance measure for the policy $\pi(\boldsymbol{\theta})$ |
| $\hat{\mathbf{s}}$ | feature-vector of the next state predicted by the model |
| $\hat{r}$ | next reward predicted by the model |
| $G_t$ | return following time $t$ |
| $\widehat{G}_t$ | approximated return by one-step lookahead, following time $t$ |
| $\widehat{G_t^a}$ | vector of returns following time $t$ with an element $G_t$ computed for all $a \in \mathcal{A}$ |
| $b(\mathbf{s}, a, \mathbf{w})$ | ZTEM TD target at time step $t$ |
| $g(s, \mathbf{w})$ | update target when acting greedily with respect to the model based on the values of ZTEM TD target |
| $k$ | number of planning steps or iterations in AVI and AAVI |
| $\delta_t$ | temporal-difference(TD) error at time $t$ |
| $n$ | number of episodes in a phase |
| $\mathcal{F}$ | model's forward transition matrix |
| $\alpha^{\mathcal{F}}$ | step-size corresponding to the matrix $\mathcal{F}$ |
| $\boldsymbol{\eta}$ | expected reward vector used by the model |

# Chapter 1

# Introduction

This chapter introduces the story of this thesis, highlights some key concepts, and provides an organizational structure of the thesis.

Imagine a world filled with intelligent assistants that help humans by understanding their questions, predicting their behaviors, reasoning about their needs, optimize their lives as a whole, and truly integrating into everything humans do. The benefits of such assistants could be enormous: from optimized global energy and resources and reducing the environmental footprint of our society, to improved psychological and physical well-being.

Intelligent assistants built on voice interaction—*conversational AI*—are among the most important applications of AI. Conversational AI assistants are already a large part of everyday human-computer interactions. Some examples are intelligent assistants that help humans with voice shopping, voice web browsing that especially helps visually impaired people accessing the internet, and virtual employee assistants. The integration of voice assistants in our lives continues to influence everyone.

Yet, we still do not know how to build fully capable intelligent assistants that can fulfill the promise of assisting people, and conversational AI assistants are no exception. Over the last several decades, there have been many attempts to build conversational AI assistants (e.g., Carbonell, 1971; Winograd, 1971; Litman et al., 2000; Singh et al., 2002). Efforts have been made not only by researchers but also by the world's largest corporations. The task of building conversational AI is so difficult that even big corporations with all the resources at their disposal are still working on it (Maedche et al., 2016; Baym et al., 2019). A great example of such efforts is an intelligent user interface for Microsoft Office nicknamed Clippy—an animated character that would pop up on the user's screen to assist and ask if the user needs help. Clippy is a great example of how a research effort seeking to develop a human-like interface and software that would assist humans results in an annoying feature that distracts the user instead.

Why is it so difficult to get to the goal of an intelligent conversational AI assistant that is truly helpful to its users and understands users' purposes? One of the reasons is that conversational AI has many complex elements that research can be focused on: dialogue management, input generation, natural language understanding, speech recognition, natural language generation (Eric, 2020). Further, each of these elements can have its own complexity levels and can result in many different research directions. For example, the complexity levels can vary from less complex basic chatbots (e.g., Quarteroni and Manandhar, 2009) that have only built-in knowledge and do not improve over time to more complex complex and advanced assistants that serve a specific purpose such as booking tickets (e.g., Handoyo et al., 2018) and can carry context from one interaction to the next which enhances the user experience.

In this thesis, I argue that to make progress in conversational AI we need a domain that is small enough for an assistant to fully understand, yet challenging enough for the

assistant to learn. I propose a voice document-editing domain that is tightly scoped and fully accessible while also exhibiting the characteristics of conversational AI agents necessary for advancing this research.

The domain itself is not enough for developing conversational AI and we further seek the most suitable methods. I further argue that, rather than using classic supervised learning approaches, reinforcement learning—and in particular more complicated model-based reinforcement learning—is well-suited for developing effective intelligent assistants. General methods, such as supervised learning, that scale with increased computation are the dominant way to build knowledge into AI systems; however, this built-in knowledge is not enough for building conversational AI agents assisting users, especially in complicated domains. Built-in knowledge does not improve over time. Supervised learning is an important kind of learning, but alone it is inadequate for learning from interaction. In contrast, reinforcement learning is a natural way to learn from human-computer interaction and allows the development of agents that can adapt quickly to their users and learn with a minimal amount of prior knowledge and, typically, in the absence of external supervision.

Reinforcement learning starts with a system—*an agent*—that is interactive and goal-seeking. Model-based reinforcement learning provides us with the key to the effectiveness of an intelligent assistant—leveraging the human-computer interaction information as much as possible. In model-based reinforcement learning an agent is augmented with a model that the agent can query to predict how the world will respond to the agent's actions—often referred to as a *world model*. The world model can further help the agent evaluate possible actions by imagining *hypothetical scenarios* and then computing their expected future outcomes—the process referred to as *planning*. Thus, world models and planning have the extraordinary potential to achieve the goal of an

efficient and useful intelligent assistant that can adapt to its users, reason about the consequences of its actions, control its choice of actions among alternatives, and learn how the real world works.

Many technologies have been explored in building conversational assistants. The challenge of building one requires for a combination of many research areas: natural language processing, human-machine interaction, dialogue systems, and deep learning, to scratch the surface. In each of these areas, many specifics would entail years of research. This thesis combines a few of them as described below.

## 1.1 Overall Structure

The thesis is divided into a background section followed by three main parts, and concluding chapters.

### Background

In Chapter 2, I formally introduce reinforcement learning settings and provide necessary definitions for reinforcement learning and model-based reinforcement learning that are common to the rest of the thesis. I leave background concepts required only for a specific chapter to be introduced within the corresponding chapter.

### Part 1. The challenge of building conversational AI

In Chapter 3, I propose a voice document-editing domain that enables the development of intelligent assistants that can become more helpful and powerful.

Then, I unfold the solution space for this domain by arguing for the need for reinforcement learning methods. I demonstrate that reinforcement learning methods are particularly well-suited for designing research and experiments when it comes to conversation AI. I carefully walk the reader through examples of how reinforcement learning fits naturally into the cycle of human-machine interaction, allowing the user to interact with a learning system in real-time, receive natural human feedback, and adjust based on the user's needs. Finally, I dive deeper into more complicated reinforcement learning methods: model-based reinforcement learning methods, in a specific setting that combines online learning and control.

## Part 2. Soft-Planner Policy Optimization

In Chapter 4, I propose a sound model-based planning method—Soft-Planner Policy Optimization (SPPO). I empirically demonstrate the effectiveness of the proposed planning method on the voice document-editing domain. Finally, I evaluate the Soft-Planner Policy Optimization's performance with respect to other state-of-the-art methods in model-based reinforcement learning.

## Part 3. Planning with Expectation Models for Control

In Chapter 5, I provide a consistent and systematic approach to planning in model-based reinforcement learning. We focus on the optimal control problem, particularly on stochastic optimal control. First, we show the incompatibility of action-value functions with expectation models. As a result, we raise the question of how planning influences the agent's action selections. We consider three strategies for this and present general model-based reinforcement learning algorithms for each. We identify the strengths and

weaknesses of these algorithms. Our algorithms and experiments are the first to treat model-based reinforcement learning with expectation models in a general setting. In Chapter 6, I apply the discussed strategies for action selection to the voice editing domain in practice. Finally, Chapter 7 provides concluding remarks and a view of the future.

I hope that this research will progress the field, starting with advancements in model-based reinforcement learning and further innovations in planning methods. These advancements will substantially improve the generalization capabilities of model-based reinforcement learning algorithms. By understanding how state-of-the-art algorithms combine planning and learning, we can better understand intelligence itself. Finding solutions for planning with model-based reinforcement learning will bring us one step closer to the goal of intelligent assistants that fully and functionally understand the real world around them. The realization of such intelligent assistants not only serves our objectives of achieving goals and ambitions of AI but also results in increasing the potential for a wide range of reinforcement learning applications that fundamentally and profoundly change the everyday life of normal people and directly benefit society: from improving productivity to benefiting people with limited abilities.

This thesis is designed to be read by an audience with some mathematical background and some familiarity with machine learning concepts. The background chapter defines terms frequently used in reinforcement learning.

# Chapter 2

# Background

The role of this chapter is to provide necessary definitions and concepts that are applicable and common across this thesis. In particular, we introduce reinforcement learning (Section 2.1), the function approximation setting (Section 2.3), and model-based reinforcement learning concepts (Section 2.5). Concepts that are applicable only for specific chapters are introduced as additional background sections in those chapters, e.g., the value iteration concept is introduced in Section 5.2.1. If a reader if familiar with reinforcement learning they may prefer to skip this chapter.

## 2.1 Reinforcement Learning

Reinforcement learning (RL) is learning to maximize the expected outcome by mapping situations to actions. RL problems are cast as sequential decision-making problems in which an agent and an environment interact in a sequence of discrete time steps $t = 0, 1, 2, 3, ..., T$. The time step at which the agent-environment interaction—an *episode*—terminates is denoted by $T$. At each time step, the environment sends a

single number to the agent called the *reward*. The goal of a reinforcement learning problem is for the agent to maximize the total reward it receives over the episode. To follow Sutton and Barto (2018), we use the terms *agent, environment*, and *action* instead of the engineering terms *controller, controlled system*, and *control signal*. Similarly, we also restrict our attention to discrete time steps.

### 2.1.1 Markov Decision Process

A mathematical framework that models reinforcement learning problems using precise theoretical statements is the Markov Decision Process (MDP). In an MDP, action selection is driven by the information that the state $s$ provides; outcomes—or next states—are partly random and partly under the control of an agent making decisions. If $\mathcal{S}$ is the set of all possible states in the given environment, then at each discrete time step $t = 0, 1, 2, 3, ..., T$ the agent receives a representation of the *environment state* denoted $S_t \in \mathcal{S}$. If $\mathcal{A}(S_t)$ is the set of all actions possible from state $S_t$ then an action that the agent takes at $S_t$ is denoted $A_t \in \mathcal{A}(S_t)$. Once an action is taken, the agent receives a numerical *reward*, $R_{t+1} \in \mathbb{R}$ and transitions to the next state $S_{t+1} \in \mathcal{S}$. The function $p$ defines the transition dynamics of the MDP such that, given particular values of the current state $s$ and action $a$, $p$ gives the probability of each possible reward $r$ and next state $s'$:

$$p(r, s'|s, a) \doteq \Pr\{R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a\}.$$

The agent interacts with the MDP yielding a sequence that we henceforth refer to as a *trajectory* denoted by $\tau$:

$$\tau \doteq S_0, A_0, R_1, S_1, ..., S_{T-1}, A_{T-1}, R_T, S_T.$$

If the probability of each possible value for $S_{t+1}$ and $R_{t+1}$ depends only on the previous state $S_t$ and action $A_t$, and not on any earlier states and actions, then the state is said to have the *Markov property*. In this thesis, we will assume that the environment state always has the Markov property.

A reward received some time in the future is worth less than if it were received immediately. This concept is known as *discounting*. A *discount rate* is the rate that indicates a willingness to trade-off between present rewards and delayed rewards. A discount rate is denoted by $\gamma \in [0, 1]$. The sum of discounted rewards is referred to as a total *discounted return*, or *return* for short, denoted $G_t$:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}. \tag{2.1}$$

When it comes to discounting there are other formulations that exist, for example, finite-horizon reinforcement learning (De Asis et al., 2019) and average-reward reinforcement learning (Schwartz, 1993). These choices are discussed in further details in Section 2.5.3.

### 2.1.2 Tabular Formulation

In this section, we describe the core concepts of reinforcement learning algorithms in a simple form: as if the states and action spaces are so small that it is possible to form arrays of tables that contain an entry for each state or a state-action pair and their corresponding policies and value functions. These are referred to as *tabular methods*.

The behavior of the agent is determined by a *policy*. If the agent follows policy $\pi_t$ at time $t$ then $\pi(a \mid s)$ is a probability distribution over all $a \in \mathcal{A}(s)$ for each $s \in \mathcal{S}$ such that $A_t = a$ if $S_t = s$. A *policy* at time $t$ is denoted as $\pi_t$. The agent aims to find a policy that maximizes the expected sum of discounted rewards $G_t$—the process is referred to as learning the *optimal policy* $\pi_*$. Generally, reinforcement learning methods specify how the agent's policy is changed as a result of the agent's experience (Sutton and Barto, 2018).

The majority of reinforcement learning algorithms involve estimating functions that compute the value of a given state or a state-action pair—referred to as *state-value* functions and *action-value* functions respectively. The state-value function, denoted $v_\pi(s)$, is the *expected return* when starting in state $s$ and following $\pi$ thereafter:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \forall s \in \mathcal{S}.$$

The action-value function is the expected total discounted return for each state-action pair, when the agent takes action $a$ from state $s$, and then follows policy $\pi$:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right].$$

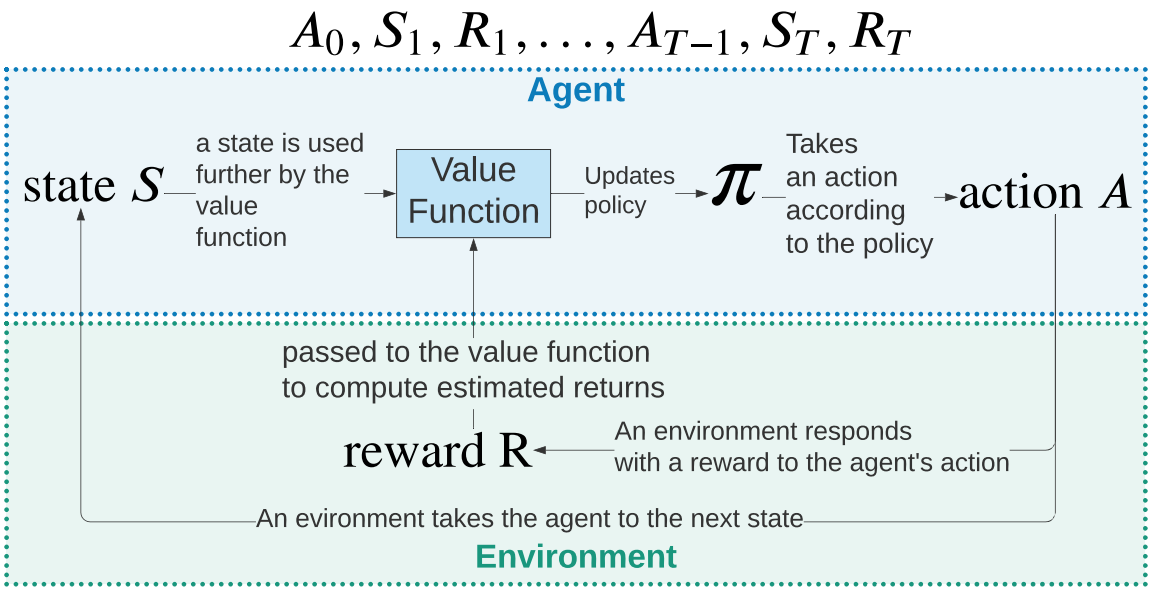


Figure 2.1: An interaction between an agent and an environment that shows policy learning.

Figure 2.1 conceptually describes the agent's policy learning. In tabular methods, it is possible to compute an optimal policy precisely.

The problem in which an optimal policy and optimal value functions are estimated is referred to as the *control problem*, which is the primary focus of reinforcement learning. Many of the methods in reinforcement learning focus on the computation of $v_\pi$ and $q_\pi$ for a fixed arbitrary policy $\pi$—referred to as the *prediction problem* (e.g., Sutton et al., 1988; Singh et al., 1995; Wan et al., 2019) or *policy evaluation*. In this work, we focus on the more challenging control problem.

A reinforcement learning agent learns within one of two broad frameworks: *model-free* reinforcement learning and *model-based* reinforcement learning (MBRL). In model-free reinforcement learning, the agent relies solely on its observations to make decisions (Sutton and Barto, 2018). In model-based reinforcement learning the agent learns

a *model* of the world, which it may further use to *plan* its decisions. The process of taking a model as input and producing or improving a policy for interacting with a modeled environment is referred to as *planning*. Models and planning are discussed in more detail in Section 2.5.

## 2.2 Partial Observability

In this thesis, we generally consider partially observable settings in which an agent receives a signal that depends on its state, but carries only partial information about the state—referred to as *observations* (Sutton and Barton, 2018, Chapter 17). These are known as Partially Observable MDPs (POMDPs).

Consider a dynamic system that takes as input at time step $t$ an action $A_t$ from a discrete set of actions $\mathcal{A}$ and emits only *observations* $O_t$. An observation does not contain all of the information for predicting the future state. Thus, the best one can do for optimal control is to rely on the full sequence of actions and observations that is referred to as *history* H. The history at time step $t$ is as follows:

$$\mathrm{H}_t \doteq A_0, O_1, A_1, O_2, ..., A_{t-1}, O_t.$$

The agent also receives a scalar numerical reward at every time step that is a function of a state: $R_t \doteq f(S_t) \in \mathbb{R}$. If $O_T$ is a special observation at the terminal state, the episode trajectory is then as follows:

$$\tau \doteq A_0, O_1, R_1, ..., A_{T-1}, O_T, R_T.$$

This trajectory is a history of what an agent interacting with the environment can know

and observe. As usual, the agent chooses actions to maximize the expected discounted return from the start state $G_t$ defined in (2.1).

### 2.2.1 Agent State

Here we define an important concept referred to as *agent state*. The agent state differs from the environment's state $s$ defined in Section 2.1. The environment state is all the things used by the environment to compute itself and is the basis of the theory of Markov decision processes. The environment state may include latent (hidden) states that are not visible to the agent. We define the agent state at time $t$ to be a compact summary of the agent's history H up to time step $t$ (including past actions, observations, and rewards). This compact summary is useful for predicting and controlling the future of the trajectory. The agent state changes at every time step based on the actions the agent takes and the observations that come from the environment. We discuss the computation of the agent state in Section 2.3 below.

In Section 2.1.1 we defined that the environment state is Markov with respect to the environment, meaning that the previous state $S_t$ carries information about all aspects of the agent-environment interaction history that allows characterizing the next state $S_{t+1}$:

$$\Pr(S_{t+1} \mid S_t) = \Pr(S_{t+1} \mid \mathrm{H}_t). \tag{2.2}$$

We denote the agent state as a bold letter $\mathbf{s}$ to differentiate it from the environment state $s$. In function approximation settings, $\mathbf{s}$ is always a feature vector, and $\mathbf{s}_t$ is a feature-vector at time step $t$. The agent state $\mathbf{s}$ is an approximation of the environment state and may no longer be Markov with respect to the environment, meaning the probability distribution of the next agent state given the full history $\mathrm{H}_t$ is no longer

equal to the probability distribution of the next agent state given the previous agent state:

$$\Pr(\mathbf{s}_{t+1} \mid \mathbf{s}_t) \neq \Pr(\mathbf{s}_{t+1} \mid \mathrm{H}_t). \tag{2.3}$$

Sometimes the agent state can be confused with the concept of belief state that is defined in partially observable MDPs. In partially observable MDPs it is common to learn a probability distribution over environment states referred to as *belief state* or *belief*. The belief state assumes that there is a set of latent (hidden) states $X_t$ and the agent is trying to learn a distribution over the latent states which is often computationally challenging. The agent has limited computational resources and sometimes cannot approximate the latent state because the latent state can be too big and complicated, especially in real-life settings. In contrast, the agent can approximate the agent state. Thus, the latent state $X_t$ produces an observation $O_t$ which the agent sees and uses to create an $\mathbf{s}_t$, but the latent state itself is never available to the agent. Unless otherwise specified, henceforth we refer to the agent state by *state* and to the environment state as *environment state*.

## 2.3 Function Approximation

In many cases, especially real-life scenarios, tabular methods do not fit the problem because they cannot account for all possible entries in a table (e.g., the number of states can be too large to in a table). In these cases, the value functions and policy are approximated using some parameterized function representation, for example, neural networks. These methods are referred to as reinforcement learning with *function approximation* or *approximate solution methods*.

In the function approximation setting, the agent state can be represented by a feature-vector that is a set of real numbers $\mathbf{s} \in \mathbb{R}^d$ that is computed using a *state-update* function $u$ (Sutton and Barto, 2018). The state-update function uses the most recent observation and action along with the most recent agent state to recursively compute the new agent state incrementally at every time step:

$$\mathbf{s}_t \doteq u(\mathbf{s}_{t-1}, A_{t-1}, O_t). \tag{2.4}$$

The state-update function $u$ may be constructed based on prior knowledge (e.g., Albus, 1971; 1981) or learned (e.g., Littman et al., 2002).

In the tabular reinforcement learning setting, we define *true* value functions with respect to environment states (see Section 2.1), conventionally denoted $v_\pi$, $v_*$, $q_\pi$, and $q_*$ (Sutton and Barto, 2018). For instance, for a policy $\pi$, the true value function $v_\pi$ is the expected return given the environment state (which is a Markov state). As described above, in the function approximation setting the agent does not have access to the environment state, nor is the agent state necessarily Markov as in (2.3). Hence the true value functions $v_\pi, v_*, q_\pi, q_*$ cannot be computed with respect to states. Instead, we have approximate value functions $\hat{v}, \hat{q}$ using deterministic functions with parameters

that are generally denoted as $\mathbf{w}, \boldsymbol{\theta}$. Their parameters are generally weight vectors where each vector is a column vector with a fixed number of real valued components, for example, $\mathbf{w} \doteq (w_1, w_2, ..., w_d)^\top$ (the $\top$ denotes transpose, to turn the horizontal row vector in the text into a vertical column vector).

The policy is often parameterized with parameter vector $\boldsymbol{\theta}$. Then an action $a$ taken at time $t$ in state $\mathbf{s}$ with parameter $\boldsymbol{\theta}$ is written as:

$$\pi(a \mid \mathbf{s}, \boldsymbol{\theta}) = \Pr\{A_t = a \mid \mathbf{s}_t = \mathbf{s}, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}.$$

In other words, we say that each time step's action $A_t$ is selected according to a policy $\pi$ mapping states to a probability distribution over actions:

$$A_t \sim \pi(\cdot \mid \mathbf{s}_t, \boldsymbol{\theta}_t), \text{ for all } t.$$

Approximated value functions are deterministic functions of weight vector $\mathbf{w}$ and are differentiable functions of $\mathbf{w}$ for all $\mathbf{s} \in \mathcal{S}$:

$$\hat{v}(\mathbf{s}, \mathbf{w}) \approx v_\pi(s),$$
$$\hat{q}(\mathbf{s}, a, \mathbf{w}) \approx q_\pi(s, a).$$

If the agent follows a policy $\pi$, then the approximate state-value function for policy $\pi$, denoted $\hat{v}_\pi : \mathbf{s} \times \mathbb{R}^d \to \mathbb{R}$, with weight vector $\mathbf{w} \in \mathbb{R}^d$, would be learned such that $\hat{v}(\mathbf{s}_t, \mathbf{w}_t) \approx G_t$, for all $t$. The quality of the approximation will be determined by the dimensionality $d$ of the weight vector, by how these vectors are used in the function approximator, and by how well they match the statistics of the environment.

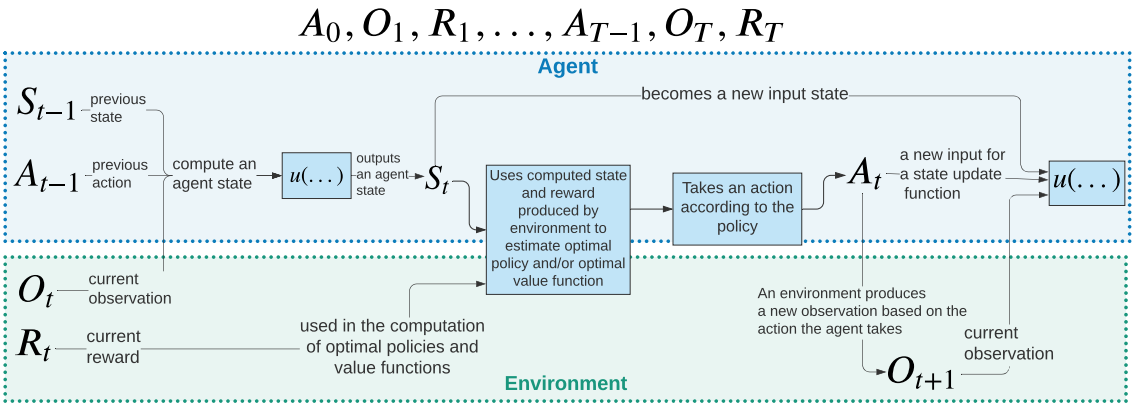


Figure 2.2: A general process in the function approximation case for updating the agent state and selecting actions.

### 2.3.1 The Agent's Components

When using function approximation, the agent is generally comprised of two primary components. The first component of the agent is the state-update function $u$ described in (2.4). The second component of the agent comprises two closely related functions: a policy and a value function. As described earlier, the value function estimates the expected upcoming reward as a function of either states or state-action pairs. In function approximation settings policies and value functions are often approximated by artificial neural networks (e.g., Parr et al., 2008; Sutton et al., 2008). A general process for updating the agent state and selecting actions using these components is shown in Figure 2.2.

### 2.3.2 Function Approximators

Reinforcement learning has been successfully applied in a function approximation setting using deep neural networks, kernel methods, support vector machines (SVMs), and all other types of function approximators. Some of these types can be more or less suited for a particular reinforcement learning problem. Neural networks (NNs) are the most popular function approximators used in reinforcement learning, and are the choice in this thesis.

## 2.4 Model-free method: Q-learning

Here we briefly describe a model-free algorithm that is used for solving control problem. In the control problem, often we are interested in approximating the action-value function $\hat{q}(\mathbf{s}, a, \mathbf{w}) \approx q_*(\mathbf{s}, a)$ where $\mathbf{w} \in \mathbb{R}^d$ is a finite-dimensional weight vector. A known control algorithm for learning action-value functions is *Q-learning* (Watkins, 1898). The update for action-value prediction in Q-learning with function approximation, step size $\alpha$, and discount rate $\gamma$ is as follows:

$$\mathbf{w}_{t+1} \doteq \mathbf{w} + \alpha \Big[ R_{t+1} + \gamma \max_a \hat{q}(\mathbf{s}', a, \mathbf{w}) - \hat{q}(\mathbf{s}', a, \mathbf{w}) \Big] \nabla q(\mathbf{s}', a, \mathbf{w}). \tag{2.5}$$

## 2.5 Model-based Reinforcement Learning

Model-based reinforcement learning allows the agent to learn faster. In the reinforcement learning methods we have discussed so far, the reinforcement learning agent learns policies and value functions. In addition to policies and value functions, the agent can learn a world model that can be used to further improve the agent's behavior. By

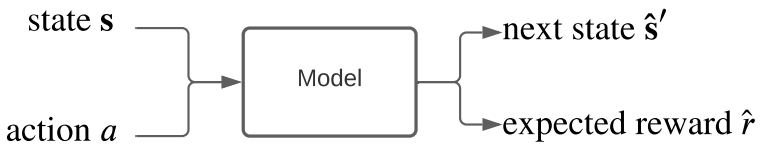


Figure 2.3: A model receives a state and an action as inputs and outputs the next state and reward.

the *world model*, we mean any function that the agent uses to predict how the environment will respond to the agent's actions. The world model is something that the agent could learn or it could be provided to the agent. The world model is an incomplete approximation of the environment.

We can use any computational process that takes the world model and produces an improved agent's behavior (via an improved policy or value function) for interacting with the modeled environment. This process is known as *planning*. The idea of augmenting a reinforcement learning agent with a world model that can be used for planning is known as *model-based reinforcement learning* (Sutton and Barto, 1981; Sutton and Pinette, 1985; Sutton, 1990; Chapman and Kaelbling, 1991; Singh, 1992; Atkeson and Santamaria, 1997; Wiering et al., 2001; Abbeel et al., 2007; Sutton et al., 2008; Ha and Schmidhuber, 2018; Holland et al., 2018; Schrittwieser et al., 2020). Henceforth we use the words 'model' and 'world model' interchangeably.

Typically, a model receives a state $\mathbf{s}$ and an action $a$ as inputs and generates the next state denoted $\hat{\mathbf{s}}'$ and reward $\hat{r}$ as shown in Figure 2.3 (Kuvayev and Sutton, 1996; Sutton et al., 2008; Hester and Stone, 2011). This output is used in planning to further improve policies and value functions.

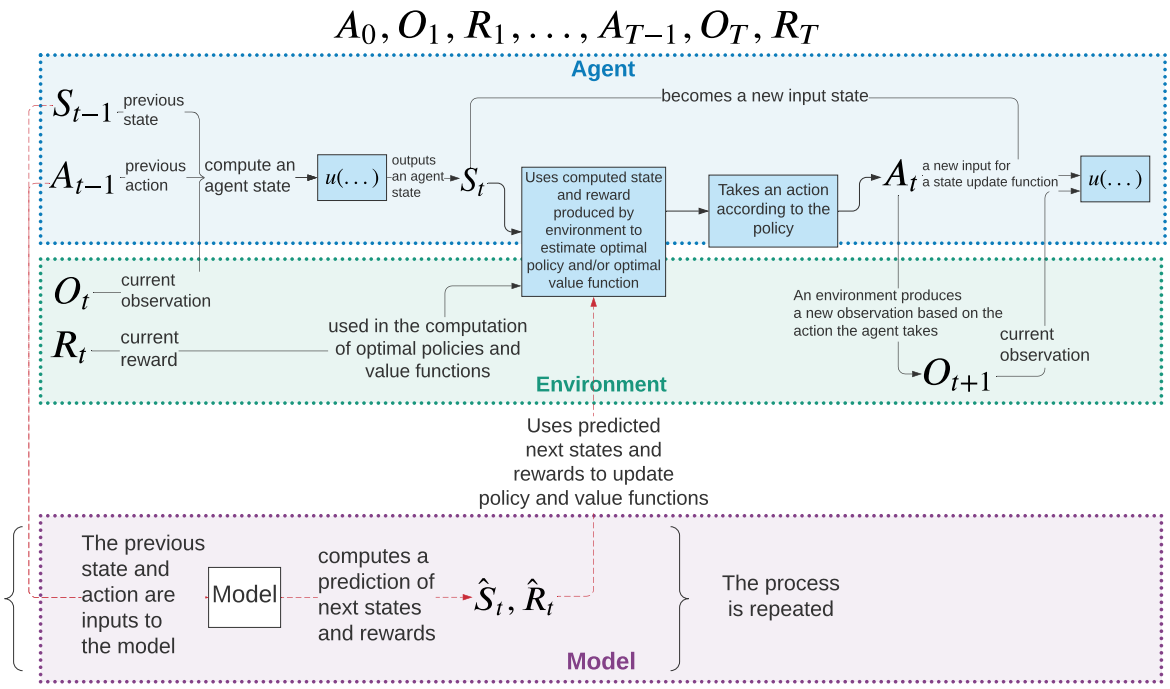


Figure 2.4: The role of model and its impact on policy and value function in a general process in the function approximation case for updating a state and an action selection.

We demonstrate the role of the model in detail in Figure 2.4—the model here is a third component in addition to the state update function and the value function components shown in Figure 2.2. Using models and planning enables an agent to predict what would happen if actions are executed from states prior to actually executing them and without necessarily being in those states.

In the tabular setting, a model takes an environment state and action as input and outputs the next environment state and reward. In the function approximation setting, while the agent state is not Markov with respect to the environment, it is Markov with respect to the model. This means we can define models that use the agent state instead of the environment state as input and then operate in the same way by producing the next agent state and reward.

We refer to the term *true model* of the environment in cases when the model is not learned, but rather is given to the agent. In this case the model has the true transition dynamics (e.g., in chess).

We now describe the four inevitable choices of model-based reinforcement learning setting:

1. Experience replay vs. learned models.
2. Model types: expectation, distribution, and sample models.
3. Episodic vs. continuing setting.
4. Linear vs. non-linear value functions.

### 2.5.1 Experience Replay vs. Learned Models

One choice to make is where planning improvements could come from: *experience replay* (ER) (Lin, 1992) or *learned models*. The trajectories that the agent has already seen are referred to as *experience*. The experience stored in the agent's memory includes the actions that the agent has already taken and the states the agent has seen. Replaying experience can be compared to how humans replay memories of the situations they have been in. In experience replay-based methods—also often referred to as *replay buffer*—the agent plans using experience stored in the agent's memory. Some replay-based implementation examples include deep Q-networks (DQN) (Mnih et al. 2013; 2015) and its variations: double-DQN (van Hasselt et al., 2016), DQN with prioritized sweeping (Schaul et al. 2016), deep deterministic policy gradient (Lillicrap et al., 2016), and rainbow DQN (Hessel et al., 2018).

Model-based reinforcement learning methods in which 1) the model is parameterized by some learnable weights, 2) the agent learns the parameters of the model, and 3) the agent then uses the model to plan an improved policy, are referred to as planning with *learned parametric models*. Learned parametric models are used in the Dyna architecture (Sutton, 1991), in normalized advantage function methods that incorporate the learned model into the Q-learning algorithm based on imagined rollouts (Gu et al., 2016), and in MuZero (Schrittwieser et al., 2020). The latter is a combination of a replay-based method and a learned model: the model is trained by using trajectories that are sampled from the replay buffer.

If the environment is non-stationary, as in many assistive systems—and in particular voice editing—then the transitions stored in the replay buffer might be stale and can slow down or even hinder the learning progress. In this thesis, we focus on learned parametric models. Learned models can lead to better generalization. Further, learned models are useful when planning forward for agent's behavior (van Hasselt et al., 2019).

### 2.5.2 Type of Models

Another important choice in model-based reinforcement learning setting is the choice of model type. A model enables an agent to predict what would happen if actions were to be executed from states prior to actually executing them and without necessarily being in those states. Given a state and action, the model can predict a sample, an expectation, or a distribution of outcomes, which results in three model-type possibilities.

The first possibility is when a model computes a probability $p$ of the next state as a result of the action taken by the agent. We refer to such a model as a *distribution*

*model*. Such models have been used typically with an assumption of a particular kind of distribution such as Gaussian (e.g., Chua et al., 2018). For example, Deisenroth and Rasmussen (2011) proposed a model-based policy search method based on probabilistic inference for learning control where a distribution model is learned using Gaussian processes. Learning a distribution can be challenging: 1) distributions are potentially large objects; and 2) distribution models may require an efficient way of representing and computing them.

The second possibility is for the model to obtain a sample of the next state rather than computing the full distribution. We refer to such a model as a *sample model*. The output of sample models is smaller than the output of distribution models and thus, more computationally feasible. In this sense, sample models are similar to experience replay. Sample models have been a good solution when a deterministic environment appears stochastic or non-stationary because of function approximation in the model. Feinberg et al. (2018) used a sample model to improve value estimates in a method called model-based value estimation. Another example is simulated policy learning (Kaiser et al., 2020) in which a variational autoencoder (Kingma and Welling, 2014) was used to model the stochasticity of the environment. A disadvantage of a sample model is an additional branching factor in planning, as multiple samples need to be drawn to gain a representative prediction.

The third possibility is to have a model that produces an expectation of the next state feature-vector instead of the probability $p$ as in distribution models. We refer to this kind of model as an *expectation model*. Expectation models have been an obvious choice for deterministic environments (Oh et al., 2015; Leibfried et al., 2017; Kurutach et al., 2018). Wan et al. (2019), Sutton et al. (2008), and Parr et al. (2008) all used linear expectation models. Wan et al. (2019) showed that expectation models can also

be used for planning in stochastic environments without a loss in planning performance when using a linear value function.

In this thesis, we focus on expectation models parameterized by learnable weights. Expectation models are easier to learn due to their compactness (Wan et al., 2019) and take away additional parameters that would be required when learning distribution or sample models. Expectation models in control settings with function approximation have been previously explored. Sorg and Singh (2010) proposed model-based planning with an expectation model and illustrated the use of linear-option expectation models compared to primitive-action linear expectation models on the continuous rooms world domain. Buckman et al. (2018) used an ensemble of expectation models and action-value functions. Jafferjee (2020) evaluated the imperfection of Dyna-style algorithms in the context of function approximation and learned models.

### 2.5.3 Episodic vs. Continuing Setting

Another choice to make is whether to use the episodic or continuing problem setting; both are typical formulation of sequential decision making problems. In continuing problems the interaction between agent and environment goes on and on forever without termination or start states. Control with function approximation in continuing problems is arguably the problem setting that matters most for AI (Sutton and Barto, 2018: Chapter 10; Naik et al., 2019). Arguably, real life settings should be formulated as continuing problems rather than problems that use discounting. However, a number of theories and concepts today are known and applicable only to episodic settings. Thus, studying episodic problems is a natural stepping-stone towards the problem of AI. Moreover, the discounted formulation is not suitable in the case of continuing control with function approximation (Naik et al., 2019) and the average reward setting

should be considered.

In this thesis, we focus on planning for control with function approximation in the *episodic setting*, extending Wan et al.'s (2019) work on the prediction problem (see Chapter 5). The episodic formulation involves termination of episodes, and thus requires an explicit indication of termination in our definitions.

### 2.5.4 Linear Value Functions

One more choice to make is between linear and non-linear value functions. In Section 2.3, we discussed how in the function approximation setting we have approximate value functions $\hat{v}, \hat{\pi}$ parameterized by learnable weights (e.g., $\hat{v}(\mathbf{s}, \mathbf{w}), \mathbf{w} \in \mathbb{R}^d$). In this thesis, we focus on linear value functions. Linear methods approximate a state-value function by the inner product between $\mathbf{w}$ and a $d$-dimensional feature vector of the agent state $\mathbf{s}$ (recall, that the $\top$ denotes transpose, needed here to turn the horizontal row vector in the text into a vertical column vector):

$$\mathbf{s} \doteq (x_1(\mathbf{s}), x_2(\mathbf{s}), ..., x_d(\mathbf{s}))^\top,$$
$$\hat{v}(\mathbf{s}, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{s} = \sum_{i=1}^{d} w_i x_i(\mathbf{s}).$$

An approximate action-value function is the inner product between $\mathbf{w}$ and a $d$-dimensional feature vector of the agent state $\mathbf{s}$ for each $a \in \mathcal{A}$:

$$\hat{q}(\mathbf{s}, a, \mathbf{w}) \doteq (\mathbf{w}^\top)^a \mathbf{s} = \sum_{i=1}^{d} (w_i)^a x_i(\mathbf{s}).$$

If both approximate state-value function and action-value function are used in the

same algorithm, we denote the weight for the action-value function as $\mathbf{w}^q$, and $\mathbf{w}$ for the state-value function.

The gradients of the approximate value functions with respect to $\mathbf{w}$ are

$$\nabla \hat{v}(\mathbf{s}, \mathbf{w}) = \nabla(\mathbf{w}^\top \mathbf{s}) = \mathbf{s},$$
$$\nabla \hat{q}(\mathbf{s}, a, \mathbf{w}) = \nabla(\mathbf{w}^\top \mathbf{s}) = \mathbf{s}.$$

One may think that the choice of linear value functions limits the expressivity of the value function. We argue that the state-update function $u$ can encompass the necessary expressivity of the features. In many cases, we can choose to have a linear value function by changing a feature representation with the $u$ function. For example, to change a feature representation, a choice for a state-update function $u$ can be variational autoencoders (e.g., Kingma and Welling, 2014; Rezende et al., 2014), long short-term memory networks (Hochreiter and Schmidhuber, 1997), predictive state representation (Littman et al., 2002), temporal difference networks (Sutton et al., 2005), or generalized value functions (White, 2015). Choices of linear representations have a long history in the field of reinforcement learning and form the basis of many methods with strong theoretical guarantees (Sorg and Singh, 2010). Further, in Section 5.3 we show that this choice allows us to use expectation models instead of distribution models without a loss in generality.

# Chapter 3

# The Challenge of Conversational AI

The role of this chapter is to introduce the problem that we use in the rest of the thesis. In my first contribution, I propose a solution to the problem, along with a particularly appropriate class of methods for solving it. The following chapters build on this first contribution. If the reader is interested only in the technical details of proposed methods, they could skip this chapter and proceed to chapters 4 and 5.

Specifically, the chapter starts with a discussion of challenges when building conversational AI assistants. As a solution, I propose voice document editing as a particularly promising domain for conducting this research. I present the proposed domain and its primary advantages—that the domain is tightly scoped and that it provides something for the conversation to be about (the document) that is delimited and fully accessible to the intelligent assistant. I further argue that model-based reinforcement learning is a particularly appropriate class of methods. Then, I discuss the advantages of reinforcement learning in general; that reinforcement learning methods are designed to learn from interaction without explicit instruction and that it formalizes the purposes of the assistant. Finally, I dive deeper into more complicated reinforcement learning

methods: model-based reinforcement learning methods, that can enable a genuine understanding of the domain of discourse and thereby can enable the assistant to work particularly efficiently with the user to achieve their goals.

## 3.1 Intelligent Assistants

The ambition of AI research is not solely to create intelligent artifacts that have the same capabilities as people; we also seek to enhance our intelligence and, in particular, to build intelligent artifacts that assist in our intellectual activities. Intelligent assistants are a central component of a long history of using computation to improve human activities, dating at least back to the pioneering work of Douglas Engelbart (1962). Early examples of intelligent assistants include sales assistants (McDermott, 1982), scheduling assistants (Fox and Smith, 1984), intelligent tutoring systems (Grignetti et al., 1975; Anderson et al., 1985), and intelligent assistants for software development and maintenance (Winograd, 1973; Kaiser et al., 1988). More recent examples of intelligent assistants are e-commerce assistants (Lu and Smith, 2007), meeting assistants (Tür et al., 2010), and systems that offer the intelligent capabilities of modern search engines (Fain and Pedersen, 2006; Thompson, 2006; Croft et al., 2010). Building intelligent assistants has been positioned as one of the key areas of development in AI (Waters, 1986).

Intelligent assistants built on voice interaction—*voice personal assistants* such as Amazon Alexa, Google Personal Assistant, Microsoft Cortana, Apple Siri, and Facebook Portal—are considered to be among the most important current applications of artificial intelligence. Voice personal assistants help people with many daily tasks, such as shopping, booking appointments, setting timers, and filtering emails. The economic

value generated by these systems demonstrates their importance for general public, and more specifically, for disabled people. The way these systems generate revenue includes selling hardware devices, such as smart speakers, and selling digital goods, such as subscription services. The revenue numbers speak for their economic value: the revenue from such voice commerce was forecast to grow from $1.6 billion in 2015 to $19.4 billion by 2023 (Tractica, 2016; Moar and Escherich, 2021). Such economic impact demonstrates that voice personal assistants are a driver of one of the important areas of AI's societal and economic impact.

## 3.2 Conversational and Purposive Assistants

Today's voice assistants are still fairly limited in their conversational abilities and we can expect more in their evolution toward increasing capability. Smart speakers and voice applications are a result of the foundational research that has come to life in today's consumer products. These systems can complete simple tasks well: send and read text messages; answer basic informational queries; set timers and calendar entries; set reminders, make lists, and do basic math calculations; control Internet-of-Things-enabled devices such as thermostats, lights, alarms, and locks; and tell pre-recorded jokes and stories (Hoy, 2018). Although voice assistants have greatly improved in the last few years, we are still looking forward to a future where assistants are capable of completing more complicated routines, such as re-scheduling appointments in a calendar, changing a reservation at a restaurant, or having a conversation.

The natural question is "Are today's voice systems 'conversational'?" The literature defines intelligent assistants conversational if they are able to recognize and respond to input; to generate their own input; to deal with conversational functions, such as turn-

taking, feedback, and repair mechanisms; and to give signals that indicate the state of the conversation (Cassell, 2000). It is the ambition of conversational AI researchers to achieve a conversational AI assistant that demonstrates these properties; communicates in free-form language; and continuously adapts to users' changing needs, the contexts the users encounter, and the dynamics of the surroundings. This conversational AI assistant would need to understand the domain within which it is assisting and to provide appropriate support, and a pleasant experience for its users. The assistant with all these properties has not been developed yet and requires more research.

Conversational AI is a primary goal along the path toward creating intelligent assistants in general. In this thesis we narrow our focus on conversational AI assistants because achieving conversational AI would lead to better general intelligent assistants—intelligent assistants that have a genuine, deeper understanding of their domains and users, helping people to achieve their goals.

Key to the effectiveness of an intelligent assistant is that it is able to understand the higher-level goals of a task when assisting its users. Hawkins (1968) defined a goal or a purpose as a future state that is brought about through instrumental control of a choice of actions among alternatives. These higher-level goals or purposes are the reasons for completing a task, and motivate and influence smaller intermediate goals. For example, if a primary goal is flying to San Francisco, then intermediate goals can be purchasing flight tickets and packing the luggage. We use the word 'purpose' to refer to the combination of the high-level context of a task and the user's goals and use the words 'agent' and 'assistant' interchangeably. A user interacting with an intelligent assistant has purposes related to the user's task. An assistant that understands users' purposes and has its own purposes is a *purposive intelligent assistant*. Everyone who has worked on intelligent assistants has recognized the importance of agents' understanding of

purposes, yet an assistant that carries this property has not been developed. Building learning systems that genuinely understand purposes has been talked about for decades (Lindgren, 1968; Sun et al., 2016; Serban et al., 2017a) and is still ahead on the AI research road map in fulfilling the promise of assisting people.

Understanding users' purposes is key not only for intelligent assistants in general but, in particular, for conversational AI agents, because purpose understanding is important for voice assistants in serving their users and adjusting to the users' unique preferences. Imagine a user who wants to have a business meeting with a client and interacts with a meeting-booking assistant. In this example, the purpose is a successful business meeting. The user's initial ask is to book a lunch reservation at a French restaurant for the purpose of the meeting, which triggers an intermediate goal: to make the reservation. A meeting-booking assistant that understands the purpose may suggest an Italian restaurant instead. This is because the assistant has information that the Italian restaurant is quieter and better suited for business meetings, even though the assistant realizes that the suggested restaurant is not a requested French type. The user may accept the intelligent assistant's suggestion, believing that it might be even *better* than their initial ask because the suggestion supports the primary purpose, the meeting. The intelligent assistant's awareness of the purpose is what motivates and drives this scenario. Another example of such a purposive intelligent assistant is a robotic arm or other manipulation device controlled by a human user (e.g., a prosthesis or an industrial robot). Take the case of a human-controlled robotic assistant picking up an object on a chessboard or pushing it; both tasks would require different motions or other situation-specific actions from the assistant. If the agent understands the context when assisting with a task, then it would know how to implement a small direct move, like advancing a pawn on the chessboard, or an indirect move, like deploying a knight,

with minimal delays and micromanagement on the user's part. The assistant creates a smoother user experience by making a better choice of actions when being purposive.

A purposive intelligent assistant improves its capabilities further when it develops goals or purposes of its own. This concept is close to an idea where agent-based adjustable autonomy is not prescribed by the user (see Maheswaran et al., 2003). An example is an iRobot Roomba that cleans a house. Roomba could set a goal of not hurting itself. This intelligent assistant not only seeks and adjusts to the user and their preferences but also develops and sets its own goals and purposes.

It is important that the user should be aware of the intelligent assistant's goals. This awareness would help the user to adjust their preferences, contributing to more harmonious interaction. This adjustment between the assistant and the user increases the assistant's capabilities resulting in a better user experience.

## 3.3 The Challenge of Conversation

Efforts to build voice assistants that learn purposes are present not only in modern dialogue systems but go back through four decades of incremental research and development (see Carbonell, 1971; Simmons and Slocum, 1972; Power, 1974; Bruce, 1975; Walker and Grosz, 1978; Cohen, 1978; Allen, 1979; Pollack et al., 1982; Grosz, 1983; Woods, 1984; Finin et al., 1986; Carberry, 1989; Moore and Paris, 1989; Smith and Hipp, 1994; Kamm, 1995). SHRDLU (Winograd, 1971) was an early breakthrough: a well-known dialogue system that responded to instructions and moved objects in a simulated world. SHRDLU was primarily a language parser that simulated understanding of purposes well.

We separate modern dialogue systems into three categories to illustrate their close-

ness to purposive intelligent assistants. The first category is *entertainment systems* that provide open-domain conversations (see Huang et al., 2020). These are chatbots and systems that bring a sense of companionship (e.g., Quarteroni and Manandhar, 2009; Nonaka et al., 2012; Higashinaka et al., 2014; Li et al., 2016a; Smith et al., 2020; Zhou et al., 2020), mostly implemented with sequence-to-sequence models (Sutskever et al., 2014) or retrieval-based methods, which select responses from an existing predefined repository. It is more difficult to develop purposive intelligent assistants in this category because the user's purpose does not result in a concrete output, and thus is even less firm or clear than in other categories. The second category is *text-based instruction systems* that are primarily represented by text-based games (e.g., He et al., 2015; Kaplan et al., 2017). The user typically interacts with these systems by typing commands. This category of dialogue systems is closer to learning purposes. An example of the work advancing in this direction is the use of natural language instructions by Goyal et al. (2019). The third category is *task-oriented systems* that help users with particular tasks (see Peng et al., 2017; Liu, 2018; Zhang et al., 2018), such as finding a product, booking a reservation, or call classification (e.g., Tür et al., 2005; Bapna et al., 2017). The task-oriented systems category is the closest to our definition of a purposive intelligent assistant, yet the delivery of the task by these systems still depends on pre-designed slots and templates (e.g., Zhao and Eskenazi, 2016; Lipton et al., 2018; Liu et al., 2018; Goyal et al., 2019; Gupta et al., 2019; Zhou et al., 2019) or a handcrafted series of commands that come with if-else conditions and rules.

Modern dialogue systems have rapidly advanced in the last few years, but it still seems to be difficult to develop conversational AI that has an ability to learn purposes. One reason is that achieving the goal of learning purposes calls for large amounts of computational power, a great deal of engineering effort to overcome architectural

challenges, and continual human-machine interaction to produce the data. The introduction of deep learning techniques combined with the massive amount of data and computational power produced amazing results with the recent system GPT-2 (Radford et al., 2019) being a prime example. GPT-2 is a confined language model that predicts the next word, given all the previous words within some text. GPT-2 is an amazing engineering effort that deserves special recognition. The limitation is that the system's answering capabilities rely on word-by-word prediction, and not on genuine understanding of a domain or users' goals. Today's dialogue systems have come a long way, yet continue to remain somewhat scripted, often using a limited number of pre-defined slots and canned responses.

Why does it seem to be so difficult to get to the goal that would advance today's research community's answers to the challenges of conversational AI and to develop agents that learn purposes? One reason is that general methods may not be sufficient. By general methods we mean methods of learning functions that map an input to an output based on input-output examples—known as supervised learning. General methods that scale with increased computation continue to build knowledge into our agents; however, this built-in knowledge is not enough when it comes to conversational AI agents assisting users, especially in complicated domains.

Another reason why advancing conversational AI seems difficult is that we have high expectations of conversational agents. The exaggerated expectations lead to choices of domains with unlimited and open-ended conversation topics. In a general conversation, an intelligent assistant is expected to know everything that a human conversational partner might know and, in some cases, also specialized user-related or world-related information. For example, the agent is expected to know about relevant aspects or patterns in the wider environment and users' lives, such as what it means to have a

schedule. In other words, the agent is expected to know about the world of its user, and a user can potentially ask the agent anything.

The research community has an ultimate goal to have intelligent assistants that are able to provide support as effectively as expert human assistants can, but high expectations with respect to an agent's knowledge about the world and the users are an obstacle on the path to getting there.

This leads to the question: Are there domains in which assistants can have focused conversations with their users and be helpful without knowing everything about the rest of the world and being able to learn what they have to assist with? We argue that a conversational domain should be *tightly scoped and fully accessible* to the intelligent assistant, so that the assistant can learn to understand it. We now propose such a domain that allows us to accelerate and advance this field without immediately solving the grand challenge of human-level AI.

## 3.4 Voice Document-Editing Domain

The challenge of genuine understanding in conversational AI requires us to pick a domain that is small enough for an assistant to fully understand. The domain should be tightly scoped and fully accessible while also exhibiting the characteristics of conversational AI agents suggested by Cassell (2020). These characteristics are

1. an ability to recognize and respond to input,
2. an ability to generate its own input,
3. an ability to deal with conversational functions such as turn-taking and feedback, and

4. an ability to indicate the state of the conversation.

Selecting such a domain and making progress with it are concrete steps forward in the direction of developing conversational AI, which, in turn, contributes to a development of intelligent assistants in general.

Document editing is one of the domains that is tractable, allows an agent to focus a conversation, and displays these four characteristics. Imagine an intelligent assistant that helps create and modify a document via a free-form language conversation with a user. This conversation is focused on the document the assistant and the user are authoring. We call this domain *voice document editing* (VDE) and propose it as particularly well-suited to develop conversational AI.

The voice document-editing domain fits well into the idea we described earlier: learning purposes enables intelligent assistants to become more helpful and powerful. Thus, the advancements in this domain domain would contribute to our overall goal of developing purposive intelligent assistants. Document editing assistants could provide better help if they could understand users' purposes and allow interactions in a form that does not require predefined commands, making the interaction process timely and efficient. They could help create and edit text messages, emails, and other documents on-the-go.

Voice document-editing assistants could perform actions such as deleting and inserting words; creating and editing itemized lists; changing the order of words, paragraphs, or sentences; converting one tense to another; or fine-tuning the style. For example, if a user asks an assistant "Please move the second paragraph above," and then says "Delete the last word in the first sentence," then the assistant would know that the first sentence the user is referring to is in that particular paragraph that was moved by the assistant's previous action. A document-editing assistant could recognize the user's

input which includes the text of the document and the user's speech. The assistant could respond to the user by asking clarifying questions. The assistant could generate its own input by making text modifications. The user's reaction to a text modification would represent feedback. The state of the conversation would be then reflected as a state of a document.

Figure 3.1 demonstrates an example of a document-editing scenario with an intelligent assistant. Such assistants have been emerging for more than three decades (Ades and Swinehart, 1986; Douglas, 1999; Lucas et al. 2004).

We separate today's dictation systems into two types: the ones that allow voice text modifications *in addition to dictation* and the ones that do not. Examples of the latter type include systems such as ListNote, and the Speech Recogniser of iOS (Duffy, 2018). Our focus is only on the former type: in these systems, users can write by dictating while walking, cooking, or doing other things, and then *edit* the document by using pre-defined commands. We refer to them as *voice editing-enabled* systems. These are dictation software systems such as Dragon by Nuance (Nuance,

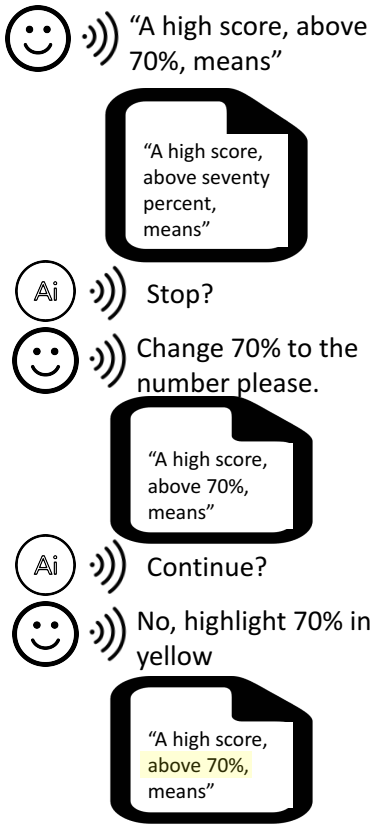


Figure 3.1: An example of a document-editing scenario with an intelligent assistant.

2003), SMARTedit (Lau et al., 2001), Apple Dictation (Gruber and Clark, 2017), Dictation.io (Digital Inspiration, 2020), Google Docs Voice Editor (Douglas, 1999; Google, 2020), and Windows Speech Recognition (Microsoft, 2020).

Document editing can be thought of as a manipulation of the manuscript in text blocks. Card et al. (1980) argued that, from a cognitive perspective, document editing is structured into a sequence of almost independent unit tasks. The unit tasks are manipulations of selected text blocks, as suggested by a number of patents related to users' text editing (e.g., Greyson et al., 1997; Takahashi, 2001; Walker, 1998). In particular, a block of text is first identified by a user. Next, the block may be moved around, modified, or formatted in place. A modification operation may include insertion of the new text, or the selected block may be removed completely. Today's aforementioned voice editing-enabled systems (Douglas, 1999; Lau et al., 2001; Gruber and Clark, 2017; Google, 2020; Microsoft, 2020) classify document-modification unit tasks into text-editing functions representing the types of text-editing operations people do in their editor of choice.

### 3.4.1 Advantages of Voice Document-Editing Domain

To demonstrate the suitability of voice document editing for conversational AI, we further look into its advantages. One advantage of a voice document-editing domain is that it excludes the real-world complexities that many other assistive systems have. Consider an intelligent cleaning robot or some other assistive robotic system (e.g., Dario et al., 1996; Salichs et al., 2019). These systems have many real-world complexities, such as the effects of the electronic hardware, materials, sensory systems, surrounding objects, and the variability of sensors. Voice document editing does not have these

dependencies and its reduced complexity is favorable for the agent's learning process and for the researchers' experimentation.

An important advantage is that the voice document-editing domain serves as a micro world with a finite number of clearly defined concepts for the agent to learn. We refer to this property of the domain as being *tightly scoped*. The world is represented by a manuscript being dictated and edited by the user. This world is smaller and more manageable than in many other real-life applications of conversational AI, such as open-domain conversations for chatbots (e.g., Saleh et al., 2020). It is easier for the agent to learn in this smaller world because the agent only has to learn about the state of the document and its modifications, both of which are fully accessible to the agent. The agent should know about the structure of the document and can learn about text blocks, such as paragraphs, sentences, and words. The agent can also know the grammatical structure and core organizational components of the document, such as salutations and valedictions when composing an email. The voice document-editing domain allows the agent to center what a conversation is about—the document itself and its edits. At the same time, the agent is not expected to be a subject-matter expert. For example, it is not necessary for the agent to have to understand history if a user is writing a historical article, or to understand medical decisions if the user is writing a health-related article. The agent is not expected to know information about the user that extends beyond the document-editing context: what the user had for breakfast, their religion, other aspects of the user's life, or additional world-related information. The voice document-editing domain allows the agent to center what a conversation is about—the document itself and its edits. The finite amount of editing concepts in the fully-accessible document defines fixed bounds for the domain and makes it easier to evaluate the performance. As a result, relative to other real-life

applications of conversational AI, the agent has to learn a smaller number of things, which is favorable for the agent and leads to reasonable expectations.

One way to evaluate our choice of voice document-editing domain is to compare it to other domains that may carry similar properties of reduced real-world complexity and being tightly scoped. We compare our domain to voice image editing and task-oriented systems: both are the closest to our definition of the purposive intelligent assistant. We show in which ways voice image editing and task-oriented systems differ from the domain of our choice, consequently making these domains more challenging than voice document editing.

A voice image editing assistant has to be able to recognize the content, which may require the assistant to learn an unlimited number of representations before becoming useful to its users. Consider a conversational image editing system proposed by Manuvinakurike et al. (2018) that is able to recognize voice commands such as "remove the tree." The ability to execute such commands entails that the system should be able not only to understand the command itself but also to have a representation of a tree to be able to identify a tree in the image that is being edited. To recognize a tree in an arbitrary image, the agent would have to learn what all possible trees look like. Learning about all possible trees requires the assistant to learn an unlimited number of tree representations, which results in the agent having to acquire infinite multi-domain knowledge. In contrast, in the voice document-editing domain, the agent does not need to understand the content of the document; it only needs to learn how to perform the edits. For example, the user dictated a phrase "There was a tree." When the user asks to replace the word 'tree' with the word 'lake,' the voice editing assistant does not need to know what the concept of a *tree* or a *lake* is. It simply needs to transcribe the word from voice to text and to know the replacement operation, because it already knows

the location of words in the text. This example illustrates that voice image editing is not tightly scoped compared to voice document editing. While conversational image editing is a suitable domain for incremental dialogue processing, it is more difficult for the agent to become useful to its users in this domain because of the large amount of information it has to learn.

Task-oriented systems also have unlimited concepts for the assistant to learn despite the focus on a particular user goal. Consider a restaurant booking system (e.g., Wen et al., 2017). To be a good assistant, the system has to know something about the user's schedule, transportation logistics, and many additional concepts. For example, the assistant needs to understand that the reservation cannot be made during the time when the user is picking up their children. The assistant also needs to know how to select the best location of the restaurant when it comes to transportation logistics. Scheduling and logistics are only a couple of many concept examples that the agent is expected to know about in addition to knowing the reservation action. The complexity of the real world leads to a possibly unlimited number of such concepts. This example demonstrates how task-oriented systems may *appear* tightly scoped while having unlimited concepts that the assistant has to learn. In contrast, our choice of voice document-editing domain makes learning possible because it is more feasible for the agent to learn a limited number of concepts and can still be useful to the user.

### 3.4.2 Current State of Voice Document-Editing Systems

Now that we have looked at the advantages of voice document editing, we turn our attention to the current state of such systems. Work on document editing via voice goes back more than three decades (e.g., Ades and Swinehart, 1986). Despite the enormous effort in this direction, today's voice editing assistive systems remain rudimentary.

When it comes to text modifications using voice, there are limitations: users can format and edit by using only a few pre-defined commands such as "New line" in order to start a new line or "Go to end of paragraph" when a user wants to move the cursor. If the user says slightly modified versions of commands such as "Let's go to a new line" or "Move the cursor to the end" instead of the aforementioned pre-defined commands, then the agent may not perform the right action. These constraints limit the benefits of editing a document via voice—at the end of the day, a user has to either perform manual text manipulation using a keyboard or carefully remember all the commands to manipulate the manuscript via voice. For example, one of the advanced editors, Google Docs Voice Editor (Google, 2020), has over a hundred basic commands that a user would need to memorize or to look up while editing. Communication to the voice editing assistant in a free-form language has not been developed yet.

An assistant that can communicate to the user in a free-form language—without pre-defined command language—and be helpful to the user is one of the main challenges in conversational AI. Developing such an assistant involves a number of complex elements, that are each challenging in isolation: e.g., natural language understanding, acoustic prosody, natural language generation, response generation, and knowledge acquisition (Eric, 2020). Voice document editing combines these elements in one domain and thereby opens up research opportunities that could benefit the advancement of conversational AI.

## 3.5 Proposed Solution Space

Methods for developing intelligent assistants should seek to answer specific requirements. First, it is intuitive that such methods should create an opportunity for human-computer interaction. Second, such methods should have the ability to incorporate user's feedback to understand the user's purposes and achieve our goal of a purposive intelligent assistant. Third, such methods should allow for continual knowledge revision that would lead to continual adaptation to users.

Reinforcement learning has been pursued as a natural approach to intelligent assistants (Kozierok and Maes, 1993; Pollack et al., 2002; Pineau et al., 2003) and is particularly fitting when it comes to methods requirements for developing intelligent assistants. Reinforcement learning starts with an agent that is interactive and goal-seeking. Formally, reinforcement learning is an approach for solving optimal control problems in which a behavior is learned through repeated trial-and-error interactions between a learning system and the world the system operates in. We introduced reinforcement learning concepts in Section 2.1. Here we describe how these concepts are mapped into a concept of an intelligent assistant. The learned behavior is a *policy* defined in (2.1), a learning system is an agent, and the world the agent operates in is an *environment*. In each interaction with the user or the world, the agent takes an action and receives a *reward* which can be a positive or a negative scalar. Reinforcement learning has not been previously investigated specifically for voice document editing and it is the approach that we are going to explore further in this thesis.

For voice document-editing assistants in particular, reinforcement learning is a natural fit. Imagine a document that could be any text of any length; for example, an email, a manuscript, or a text message. At any point in time, the user could have

two options to interact with the assistant: 1) to dictate a new block of text, or 2) to ask the assistant to modify an existing text; for example, to switch some words or sentences, delete words, or highlight parts of the text, etc. In reinforcement learning terminology, an *agent* is the document-editing assistant. An *environment* combines both a document and the user that interacts with the agent. A *state* encompasses the current document and all the user communication to the assistant. In such, the current document would include the text that was already dictated by the user, and the text's structure, such as paragraphs, sentences, and words.

At every time step a state is observed by the agent. The state includes the user's communication to which the agent then would react to and take an action: to edit the document or to request a clarification from the user. This action would then affect the environment, potentially resulting in changes to the document in the case of editing actions. The user would respond in turn to the agent's action by replying with the next request or by continuing to request edits. This response would create a new state for the agent and this interaction would repeat until the user would be fully satisfied. The satisfaction of the user would be reflected in a reward signal that the agent receives after every selected action. We leave further details of reinforcement learning formulation for voice document editing and describe such voice document-editing assistants later in this chapter.

### 3.5.1 Advantages of Reinforcement Learning

The first advantage of reinforcement learning for intelligent assistants is that it provides an opportunity for an intuitive human-computer interaction, in contrast to more common machine learning formulations of supervised and unsupervised learning. In particular, an interactive reinforcement learning agent can directly learn things about

its environment with every action and select future actions according to that knowledge. This means that agents are not learning from the input-output pairs that were provided ahead of time, but from direct experience—known as *online* learning. Online learning provides a natural opportunity for intuitive interactions, during which intelligent assistants adapt to users. Imagine a voice assistant that recommends fun things to do during travel, similar to the NJFun system (Litman et al., 2000; Singh et al., 2002) that provided users with information about fun things to do in New Jersey. The assistant can learn about the user's preferences much faster and provide better recommendations by extracting a reward signal after each suggestion is made and adjusting its suggestions accordingly. When this assistant helps a traveler who is interested in music history, it can learn the user's preferences quickly, based on how satisfied the user was with its previously provided recommendations. The assistant can tailor the recommended places to music history museums, music history festivals, music exhibits, and other similar attractions. This adjustment via online learning between the assistant and the user improves the assistant's reasoning about its future actions. The importance of online learning and interaction feedback has appeared in many studies (e.g., Long, 1981; Pica et al.; Pica, 1986; 1987; Gass and Varonis, 1994; Bassiri, 2011; Gašić et al., 2011; Gašić et al., 2013a; Ferreira and Lefèvre, 2015; Li et al., 2017a; Liu et al., 2017).

The second advantage of reinforcement learning is that users' feedback can be formulated as a reward signal. Feedback can be used as a goal-directed signal of users' satisfaction or dissatisfaction with actions the assistant takes, which is one way to evaluate the assistant (see Jiang et al., 2015). In the context of dialogue settings, it is expected that users have a way of communicating their feedback to the assistant, either through voice interaction or physical interaction via a robotic device. Examples of

these interactions include training assistants with animal-like robot clicker techniques (Kaplan et al., 2002); using a combination of reward signals from a human and an environment (Knox and Stone, 2012); and using a sparse human-delivered training signal, as in the case of adaptable, intelligent artificial limbs (Pilarski et al., 2011). An intelligent assistant maximizing users' satisfaction is a reinforcement learning agent maximizing the expected return (see 2.1).

The third advantage of reinforcement learning is that it allows changes in the agent's knowledge. These changes are fundamental for learning, which is different from built-in knowledge. A process when the agent has to change what it knows, rather than knowing whether something is a fact, is defined as *learning* by Selfridge (1993) and is considered the most important part of intelligence (Woodrow, 1946). Knowing is simply factual: one learns a particular kind of knowledge and knows if it is the truth that applies to a particular setting. Learning, however, leads to understanding. Understanding is more fluid than drawing knowledge from built-in facts: in understanding, we relate the facts to everything else and can reason about the consequences of our actions. Reasoning about the outcome results in a controlled choice of actions when presented with alternatives. As we argued earlier, a controlled choice of actions is a quality of a purposive intelligent assistant that continuously adapts to users' changing needs. Thus, reasoning about the consequences of actions leads to an assistant that learns how to learn, adapt, and improve by interacting with users (e.g., Li et al., 2017). The most brilliantly-engineered slot-filling systems will not learn purposes because they cannot learn to avoid repetitive mistakes, nor can they learn to adapt to different users. In contrast to learning from static datasets, reinforcement learning is more general and allows for these adjustments. In particular, the structure and other properties of the world are not assumed in reinforcement learning.

### 3.5.2 Prior Work Applying RL to Conversational AI

Reinforcement learning has been applied to conversational AI in various ways since the late 1970s. Some of the earliest works were Walker and Grosz (1978); Biermann and Long (1996); Levin et al. (1997); Singh et al. (2000). In more recent works, Gašić et al. (2011, 2013a) use reinforcement learning in online settings to directly learn from human interactions using rewards provided by users and to optimize the agent's behavior in reaction to the user. Dhingra et al. (2017) also explore online learning, but in simulated settings. They train their agent entirely from the feedback that mimics the behavior of real users. Such simulated settings are not always available for a learning task and building simulators for dialogue scenarios and tasks is challenging (Cuayáhuitl et al., 2005; Li et al., 2016b). To overcome these challenges, Zhou et al. (2017) choose to optimize a policy in offline settings using the raw transcripts of the dialogues, while Liu and Lane (2017a) take an approach of jointly optimizing the dialogue agent and the user simulator. Xu et al. (2018) also apply joint modeling of dialogue act selection but use reinforcement learning only to optimize response generation. A number of others also use reinforcement learning for open-domain dialogue generation (e.g., Ranzato et al., 2015; Li et al., 2016a; Yu et al., 2017; Budzianowski et al., 2017; Jaques et al., 2019).

A big trend in the last five years has been to use neural networks with multiple layers—*deep learning* —in conjunction with reinforcement learning. This line of work was invigorated and the attention was drawn to it after the success of deep reinforcement learning on games such as Atari, Go, chess and shogi (Mnih et al., 2013; Silver et al., 2018; Schrittwieser et al., 2020). This trend was embodied in several threads each pursuing a different approach. For example, in one approach Liu et al. (2017), Shen et al. (2017), Su et al. (2017) use reinforcement learning in combination with

supervised learning. Shah et al. (2016; 2018) contrast the interpretation of the human feedback as a reward value (see Thomaz et al., 2005, 2006; Knox and Stone, 2012; Loftin et al., 2014) and propose an interactive reinforcement learning approach in which the user feedback is treated as a label on the specific action taken by the agent similar to Griffith et al. (2013). In another approach, reinforcement learning is studied for policy adaptation between domains in multi-domain settings (e.g., Gašić et al., 2013b; Cuayáhuitll et al., 2016, 2017; Gašić et al., 2017; Rastogi et al., 2017; Chen et al., 2018; Liu and Lane, 2017b). Serban et al. (2017b) apply reinforcement learning to select from a number of responses produced by an ensemble of so-called response models. Tang et al. (2018) use reinforcement learning to train a multi-level policy that allows agents to accomplish subgoals. Weisz et al. (2018) focus on large action spaces. Peng et al. (2017) propose a hierarchical deep reinforcement learning approach using options (Sutton et al., 1999) for a task-oriented dialogue agent. Others apply reinforcement learning to teaching agents to communicate with each other in multi-agent environments (e.g., Foerster et al., 2016; Sukhbaatar et al. 2016; Lazaridou et al., 2016; Mordatch and Abbeel, 2018; and Papangelis et al., 2020). Bordes et al. (2017), Dhingra et al. (2017), Lewis et al. (2017), Wen et al. (2017), Williams et al. (2017), Liu et al. (2017, 2018), Jiang et al. (2019) all explore end-to-end dialog system approaches. These works are a small fraction of the large body of research that implements deep reinforcement learning approaches in dialogue systems.

A smaller trend in the last couple of years has been to use a more advanced form of reinforcement learning in which a model of the world is learned in addition to a policy and value functions. These model-based reinforcement learning methods are the topic of the next section.

### 3.5.3 Model-Based Reinforcement Learning Assistants

World models and planning are helpful in learning the optimal agent's behavior. One advantage of models and planning is that they are useful when the agent faces unfamiliar or novel situations—when the agent may have to consider actions that they have not experienced or seen before and evaluate the consequences of those actions. Models can be fundamental in making long-term predictions and evaluating the consequences of actions based on models' predictions. Planning can help the agent evaluate possible actions by evaluating *hypothetical scenarios* that are produced by the model and then computing their expected future outcomes (Doll et al., 2012; Ha and Schmidhuber, 2018). These outcomes can be computed a few steps ahead and can be thought of as the agent reasoning about the long-term consequences of its actions, similar to how people evaluate the long-term consequences of their decisions. The agent reasoning about the consequences of its actions and acting based on the world model's predictions is analogous to the way a person reasons and acts based on their understanding of the world. Hypothetical scenarios allow the agent to safely explore the possible consequences of actions. For example, the agent can use hypothetical scenarios to explore actions that in real-life applications could lead to a costly crash or another disaster (Berkenkamp et al., 2017). Another advantage of world models and planning is that they help accelerate the learning of policies (e.g., Freeman et al., 2019).

The advantages of world models and planning arise in particular with intelligent assistants. World models and planning enable intelligent assistants to leverage the interaction information as much as possible, with a minimal amount of prior knowledge and in the absence of external supervision. Nortmann et al. (2015) suggested people's internal models of the world are learned during the interaction with the world and contain the information that is perceived by their brains. Similarly, world models are

learned by the agent during human-computer interaction. Forrester (1971) described people's models of the world as the image of the world that humans carry in their head: the concepts and relationship between them that are used to represent a real system. Likewise, world models capture the dynamics of the environment and the user-agent interaction within it (Sutton, 2019). These dynamics are everything that the agent needs to know about the environment. More precisely, in the context of model-based reinforcement learning the knowledge of the environment dynamics enables the agent to predict the next states and rewards from the previous state and its actions. Thus, models help the agent to make informed action choices by enabling it to compute hypothetical future outcomes of different actions with the modeled environment. Planning is particularly important because it allows the agent to learn not only from the past actions that resulted in a reward but also from hypothetical actions for which the agent has not seen a reward. For example, Ng et al. (2004) proposed a successful application of reinforcement learning that uses a model for autonomous helicopter flight, saving hours of adjustments and flight testing and preventing unnecessary crashes. Ng et al. started with a human pilot flying the helicopter for several minutes and then used the data to fit a model of the helicopter's dynamics. Similarly, Coates et al. (2017) provided an agent with the pre-trained model and the reward function.

Thus, in conversational AI, models and planning make it possible for the agent to acquire the full benefits of reinforcement learning, which in turn would allow us to create a purposive intelligent assistant (see Section 3.2) that is efficient and useful, can adapt to its users, reason about the consequences of its actions, can control its choice of actions among alternatives, and can learn how the real world works. Planning also has been previously explored in dialogue systems outside of reinforcement learning (e.g., Stent et al., 2004; Jiang et al., 2019).

There are several prior works in conversational AI that have come close to doing model-based reinforcement learning. Most of these have used experience replay (see definition in Section 2.5) with a replay buffer (Lin, 1992; Mnih et al., 2013; 2015). For example, Lewis et al. (2017; Yarats and Lewis, 2017) explored a form of planning by using a dataset obtained through Mechanical Turk (see Paolacci et al., 2010) to generate a simulation of complete dialogues—dialogue rollouts—that are used for planning. Peng et al. (2018b) introduced a Deep Dyna-Q architecture (as in Sutton, 1991) using deep learning and a replay buffer, which was subsequently extended by Su et al. (2018) and Wu et al. (2019; see also Gao et al., 2019; Zhang et al., 2020). Zhao et al. (2020) brought this line of work the closest to the kind of MBRL for conversational AI that we are advocating in this thesis (cf. model-based Bayesian reinforcement learning, Lison, 2013) while still being limited by its reliance on a replay buffer.

### 3.5.4 MBRL Open Research Areas

There is a diversity of open research areas in model-based reinforcement learning. One area is in the direction of the choice of models (see Chua et al., 2018), such as probabilistic transition models that use Gaussian processes (e.g., Kocijan et al., 2004; Deisenroth et al., 2013; Kamthe and Deisenroth, 2018), nonlinear neural network models (e.g., Hernandaz and Arkun, 1990), generative models (e.g., Buesing et al., 2018), latent state-space models (e.g., Wahlström et al., 2015; Watter et al., 2015), and policy search methods (e.g., Bagnell and Schneider, 2001; Deisenroth and Rasmussen, 2011; Levine and Abbeel, 2014; Levine et al., 2016). Another area of study is on the asymptotic performance of model-based methods to match the asymptotic performance of model-free algorithms (e.g., Chua et al., 2018). The effectiveness of planning methods and data sample efficiency is one more area of active research (e.g., Hafner et al.,

2018; Kamthe and Deisenroth, 2018; Kaiser et al., 2020). Silver et al. (2017a) argue for the effectiveness of planning methods in applications such as AlphaGo, offering an algorithm based solely on reinforcement learning, without human data, guidance, or domain knowledge beyond game rules. Nevertheless, cases of full model-based reinforcement learning similar to Silver et al. (2017b), in which the model is learned from online data and then used for planning, are rare, especially in stochastic domains. Reduction of errors in learned models that resemble the real environment, data efficiency, asymptotic performance, and a choice of planning methods are some of the topics at the forefront of model-based reinforcement learning research.

## 3.6 Realizing Voice Document-Editing Domain

The realization of voice document-editing assistants can be naturally mapped into MBRL components: a state update function, a policy and value functions, and a model (also see Section 2.3).

The first component of a MBRL agent, a state update function, produces the agent state which would be a representation of the document and the history of the conversation and its edits. The agent's state should contain information about the current document structure, such as paragraphs, sentences, and words; the user's communications; which actions were taken by the agent; and how satisfied the user was in response to the actions taken. An agent state can be computed by a state-update function using function approximation as in (2.4); for example, a recurrent neural network such as a two-layered bi-directional Gated Recurrent Units (GRU) (Cho et al., 2014) that takes an observation as one of its inputs can be used as a state-update function. The observation can be represented by a combination of the current document's content

and the user's speech. The observation can be thought of as *some* information about what is going on in the environment—the document and the user interacting with the assistant. The information contained in the observation is partial because it does not include things such as the user's emotions or anything else outside of the document. Outside information like users' emotional reactions can be approximated from additional data such as the time between users' reactions, their tone of voice, or their facial expressions (e.g., via techniques such as face valuing by Veeriah et al., 2016, or other cues as suggested by Skantze, 2016). One choice to represent an observation could be word embeddings such as `word2vec` (Mikolov et al., 2013) or Bidirectional Encoder Representations from Transformer (BERT) (Devlin et al., 2018).

In the second component of a MBRL agent, the policy in voice document editing is the agent's behavior in response to user's requests. This behavior allows the agent to make a choice between continuing to listen to the user when the user wishes to dictate more, and selecting an editing action to manipulate a text block when the user requests an edit. A value function in voice document editing represents user's satisfaction.

The third component of a MBRL agent, a world model, helps to develop agent's understanding of what happens to the document if it takes an unseen action. The model captures the dynamics of the user-agent interactions that the agent learns by observing the results of actions taken. Learning world models in the voice document-editing domain is possible because of the domain being tightly scoped. Even if the assistant has never heard a command that the user is saying, by using environmental models and planning, the agent would recognize whether the command is a continuation of dictation or an edit that the user wants to perform. When editing requests are recognized, then the agent can plan which editorial action leads to a better outcome and higher users' satisfaction.

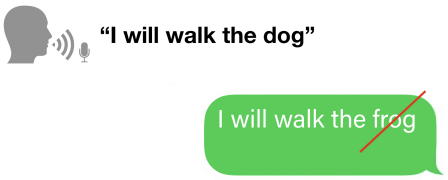


Figure 3.2: An example when a human says “I will walk the dog”, yet the dialogue system results in “I will walk the frog”. As a result, the user wishes to delete the final corrupted part of the sentence and then re-dictate it. Note that, in this example, we could have replaced the word “frog” with the word “dog”, which would comprise a replacement problem that is not within the scope of this work.

### 3.6.1 Voice Document-Editing Deletion Task

Solving for voice document editing in full is an effort that would require significant resources in time and computation that would most likely be a subject of a few graduate thesis works. To reduce the scope to one graduate thesis, we split solving VDE into a few subproblems. Recall, that examples of tasks within VDE are words insertion, words deletion, fine-tuning a style, formatting, etc. In this thesis, we focus on words deletion. Specifically, the problem is that often a system’s transcription of a dictation results in a *corrupted* part of the sentence that the user wishes to *delete* and then re-dictate (see example in Figure 3.2). We refer to this problem as a *deletion task*. The idea of deletion tasks manifests from the state of current speech recognition systems: most of today’s voice-editing systems do not allow the user to correct errors via voice.

The choice of deletion task among other VDE tasks is driven by two primary reasons. The first reason is that the the deletion task can be simulated while accurately reflecting real-life scenarios of VDE which permits the development and evaluation of methods without user interaction. The second reason is that the deletion task is well

suited for demonstrating the advantages of MBRL.

We proceed with a common real-life scenario setting where corrupted words are at the end of the sentence. Consider a speech-to-text system and a user that dictates a sentence to a dictation system or a tool, the latter further referred to as an agent. The agent processes the dictation and displays it on a screen to a user. We know that some portion of the sentence has been processed incorrectly. As soon as the agent displays processed words, a user sees the sentence right away and stops dictating. At this point, the user sees which portion of the sentence has been processed correctly and which one has been processed incorrectly. Here, in real life, the user could have many means to ask the system to correct the sentence. We proceed with a simple setting in which the user says one word "Delete" if they are not satisfied with the words at the end of the sentence that is displayed on the screen during dictation. In our settings, when the user says "Delete", the word "Delete" is treated as a keyword for the agent to take an action: the action here means to identify which words were not recognized properly and to delete them. If the agent deletes too few words, the user can say "Delete" again and the process is repeated until the correct number of words is deleted and the user is satisfied. If the agent deletes too many words, the user has to repeat the accidentally deleted words before continuing dictation. The agent's goal is to identify the correct number of words to delete in the least number of steps.

Next, we formalize the deletion task setting using reinforcement learning concepts of environment, observation, state, action, episode, etc., and provide additional definitions that are used further.

1. The environment of the deletion task is comprised of text that is transcribed by a system from a user's dictation based on the user speaking to a speech-to-text system.

2. An *encoded text* is a text that has already been dictated and is shown on the screen; the encoded text at time step $t$ is denoted as $w_t$.

3. A *speech transcription* is a new sequence of words dictated by a human; the speech transcription at time step $t$ is denoted as $\psi_t$.

4. An observation $O_t$ that is available to the agent at a given time step $t$ combines an encoded text $w_t$ and speech transcription $\psi_t$.

5. *An interaction* corresponds to an episode in reinforcement learning. An interaction is an interactive process between the user and the agent. This interaction starts during any time of dictating a sequence of words at the moment when a user decides to delete some words and says "Delete" for the first time. Interaction terminates in one of the following two cases. In the first case, the agent takes an action that completes an interaction and the user is satisfied with this action from the agent, meaning the user is satisfied with the last edits of the text. In the second case, the agent takes an action that deletes too many words. In this case, the user would have to repeat the words that were previously dictated, not corrupted, but mistakenly deleted by the agent.

6. We define $m_t$ a number of actions (or steps) that the agent takes within one interaction.

7. An intent is an integer that represents the number of words that the user wishes to correct; an intent at time step $t$ is denoted as $I_t$.

8. We define $b_t$ as a number of words that were not corrupted but deleted by the agent as a mistake: it is the number of words the user has to repeat if the agent deleted too many words. See Figure 3.3.

9. An action $A_t$ selected by an agent represents a number of words to delete.

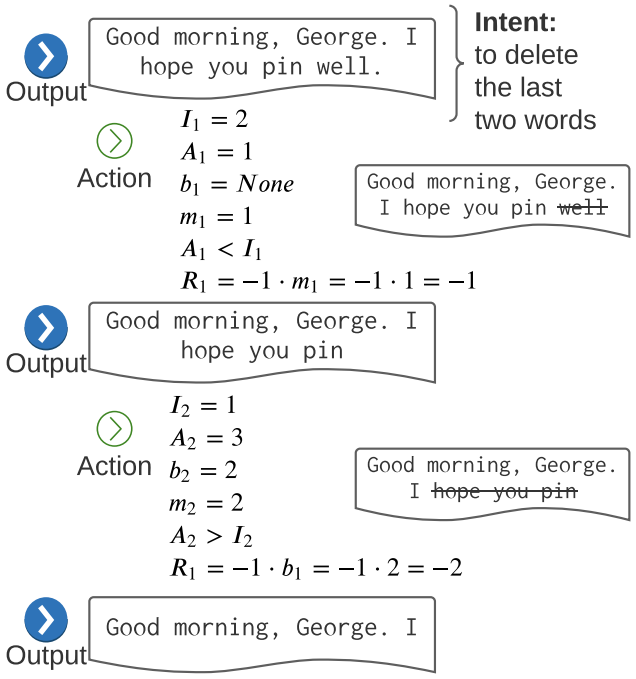


Figure 3.3: Consider an example where the user says "Good morning, George. I hope you have been well", yet the dialogue system transcribes and displays to the user "Good morning, George. I hope you pin well." The initial user's intent in this example is 2—the user desires to delete two words "pin well". If an agent takes an action $A_1 = 1$, the updated sentence would become "Good morning, George. I hope you *pin*" and the reward would be $R_1 = -1$. The updated intent would be then 1. Next, if the agent takes an action $A_2 = 3$. The updated sentence would be "Good morning, George. I" and the reward would be $R_2 = -2$. If the agent takes an action $A_2 = 3$ the user would have to re-dictate two deleted words "hope you" indicating that they were deleted incorrectly.

10. The reward $R_t$ is an integer that reflects user (dis)satisfaction by penalizing for a longer time to task completion and for overshoots that require the user to repeat the previously dictated words. The reward is 0 if the action corresponds to the intent meaning that the agent deletes exactly the right number of corrupted words

and is $-1$ for each incorrect action with an additional penalty described below. If the agent undershoots by deleting fewer words than the intent then the agent is penalized proportionally to the number of actions the agent has already taken. This way if the agent has an incentive to delete corrupted words in fewer actions. The reward is then calculated as follows:

$$R_t = -1 \cdot m_t.$$

If the action taken is greater than the intent (see the second step in the example in Figure 3.3), then the agent is penalized by the number of words $b_t$ that were deleted by the agent incorrectly. The reward then is calculated as follows:

$$R_t = -1 \cdot b_t.$$

### 3.6.2 Simulation and Dataset

The lack of sophistication in today's voice-editing systems is partially due to the difficulty of training them when it comes to the datasets. Such training requires either a large, diverse training dataset of general document-editing dialogues or hours of online human-machine interactions. Such datasets do not currently exist and online interactions are impractical— a substantial amount of training is needed before the agent can overcome users' frustration with a system that has not been sufficiently trained.

To overcome the limitations, we created a simulation of the deletion task that mimics a sentence being corrupted by the voice-recognition imperfection. In our simulation, we consider a collection of sentences as they were dictated by a user. We further inject corrupted words into a sentence simulating a corrupted end of the sentence. The

simulation is set up in a way that, followed by a corrupted sentence, the agent always receives "Delete" as a speech transcription $\psi$ from the user. Then the agent has to take an action and it keeps receiving "Delete" as an answer until the correction of the sentence is complete or until the agent overshoots the correction.

To represent a sentence collection, we chose the BookCorpus dataset that is a large collection of free novel books written by unpublished authors, which contains $11,038$ books of 16 different sub-genres (Zhu et al., 2015). The dataset contains around 74 million sentences. Along with narratives, the books sentences contain dialogue, emotion, and a wide range of interaction between characters (Kiros et al., 2015). The sentences from the dataset are fed to the agent one by one, and corrupted words are injected at a random place in a sentence. Corrupted words are chosen uniform randomly from a vocabulary of the dataset. We acknowledge that a more realistic approach would be to replace words with phonetically similar ones, but this would require significant engineering efforts such as creating datasets of phonemic dictionaries, which is beyond the scope of this thesis.

This simulation intentionally creates a difficult problem for the agent: there are no labels, and the replacements of words with corrupted ones are selected at random. Thus the only way for the agent to learn is through dialogue interactions and by observing the dynamics of the environment. Given the simulated environment, we know how many corrupted words were injected and thus can calculate rewards for any given action. Thus, our reward function well reflects a real-life human-machine interaction.

## 3.7 Summary and Implications

In this chapter, we have proposed that the domain of voice document editing is particularly well-suited for the development of intelligent assistants that can engage in a conversation. To make progress in developing useful assistants for conversational AI, these assistants should be purposive. A natural approach for developing purposive assistants is reinforcement learning, and, in particular, model-based reinforcement learning. This approach is well-suited to assistants that learn and adapt within document editing and general conversational AI settings. Many aspects of using model-based reinforcement learning remain open areas in AI research, in particular, its use within voice document editing. Finding solutions for the voice document-editing domain with model-based reinforcement learning and building these systems can provide us with lessons that move us closer to building other systems that genuinely understand the user and learn their purposes. In this way, a better voice document editing system will also contribute to the development of other assistive systems, moving the research toward the ultimate goal of assistive agents that fully and functionally understand the real world around them.

The realization of voice document-editing assistants not only serves our objectives of creating a purposive assistant and achieving goals of conversational AI, but also results in an application that directly benefits society: from improving productivity to benefiting people with limited typing abilities.

# Chapter 4

# Soft-Planner Policy Optimization

The role of this chapter is to present a second contribution of this thesis. In this chapter, I take the argument made in Chapter 3 and I apply model-based reinforcement learning methods to the proposed voice document-editing domain in practice. In particular, in this chapter I propose a novel model-based reinforcement learning method—*Soft-Planner Policy Optimization* (SPPO). The intent of this method is to make the best use of online learning and to address two problems: 1) faster performance improvement during online training, and 2) creation of an agent that is computationally light for real-life deployments.

The first problem, faster performance improvement during online training, is often referred to as *sample efficiency*. The less of an agent's experience during training is required to reach the desired behavior, the more sample-efficient that agent is. Recall that despite recent progress in voice document editing, voice-dictation systems are still in a primitive form (Section 3.4.2). One issue is that further advances in a real-life domain are limited by the availability of data due to the cost of acquiring new samples. One of the promises of model-based reinforcement learning is a substantial

improvement in sample efficiency (Sutton and Barto, 2018, Chapter 8).

The second problem, creation of an agent that is computationally light for real-life deployments, can be solved in a number of ways. Often, model-based reinforcement learning methods result in computationally heavy models that are difficult to deploy in real life. We show that our method results in a trained model-free agent that can be decoupled from a computationally heavy model and a planner. This agent can be deployed as a computationally-light final solution that performs better than a model-free agent trained without a model. We demonstrate the benefits of a model and planner with a novel soft-planner policy and show how to use a novel, non-standard update to improve the model-free policy.

In real-world human-computer interactive systems, the reward function truly depends on user feedback. The deletion task within the voice document-editing domain carries this property and encompasses user interactions with an imperfect transcription system. The agent's goal in a deletion task is to delete a certain number of words at the end of the phrase (see Section 3.6.1). We compare it to the current state-of-the-art implementations for such systems: model-free actor-critic methods.

## 4.1 Related Work

The first suggestions of formalizing MDPs for dialogue goes back forty years, followed by work using reinforcement learning (Walker and Grosz, 1978; Biermann and Long, 1996; Levin, Pieraccini, and Eckert, 1997). An example of an early end-to-end dialogue system that used reinforcement learning is the RLDS software tool (Singh et al., 2000) at AT&T Labs. NJFun was another pioneering system developed by Litman et al. (2000) that used reinforcement learning. NJFun was created as an MDP with a

manually designed state space for the dialogue.

Dialogue systems have rapidly advanced in performance over the last few years, following the introduction of a paradigm that turns one sequence into another sequence by Sutskever, Vinyals, and Le (2014)—known as *sequence-to-sequence*. Sequence-to-sequence transformed many natural language processing tasks allowing minimal assumptions on the sequence structure. Further, much of the progress in dialogue systems is now attributed to the increasing availability of large amounts of data and computational power via deep learning techniques combined with reinforcement learning—*deep reinforcement learning* (see Section 3.5.2) (e.g., Cuayáhuitl et al., 2016; Budzianowski et al., 2017; Liu, 2018; Tang et al., 2018; Mendez et al., 2019; Shin et al., 2019).

Many recent dialogue systems are based on policy gradient methods, such as actor-critic methods (e.g., Su et al. 2017; Peng et al., 2018a); however, such methods do not include models and planning that are critical components of model-based reinforcement learning allowing an agent to learn the dynamics of a dialogue environment. Several works combine policy gradient methods and supervised learning methods (e.g., He et al., 2015; Li et al., 2016a; Kaplan, Sauer, and Sosa, 2017; Weisz et al., 2018), or take a different approach to improve sample efficiency using imitation learning (Lipton et al., 2018). Zhao and Eskenazi (2016) used an end-to-end model-based reinforcement learning architecture that is close to the Dyna architecture (Sutton, 1991); however, their additional model employs a known transition function from a dataset and not a world model that is learned through interactions. Their method is comparable to Model-Based Policy Optimization (MBPO) (Janner et al., 2019) that also uses a dataset. In contrast, our method focuses on *online learning* (see Section 3.5.1) and allows the agent to learn a model online from interactions while reducing the need for additional elements that are often required in dialogue system implementations (e.g., "successful

trajectories", Lipton et al., 2018; or additional data augmentation, Goyal, Niekum, and Mooney, 2019).

A developing direction of research on dialogue systems is applying model-based reinforcement learning approaches that allow learning a world model to mimic user responses generating hypothetical experiences and to use planning to improve the agent's behavior. We discussed some of these works in Section 3.5.3. For example, Lewis et al. (2017) used dialogue rollouts, followed by Yarats and Lewis (2017) who improved the effectiveness of planning with dialogue rollouts further. Peng et al. (2018b) proposed the deep Dyna-Q framework incorporating planning into dialogue policy. Su et al. (2018) extended this work and added control of the quality of hypothetical experiences improving the planning phase. Wu et al. (2019) extended the deep Dyna-Q framework by integrating a "switcher" that allowed the agent to differentiate between a real or hypothetical experience. These works, however, do not combine learning and planning together, similar to what has been proposed by Sutton (1990, 1991). Zhao et al. (2020) used similar model designs and took the first step in combining learning and planning, bringing this line of work closer to a combination of learning, planning, and reactive execution that has been advocated for in this thesis (see Chapter 3).

## 4.2 Background

In this chapter, we focus on reinforcement learning methods that learn a parameterized policy that enables actions to be taken instead of learning action values and then using them to select actions. A parameterized policy with policy parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ is defined as follows:

$$\pi(a \mid \mathbf{s}, \boldsymbol{\theta}) \doteq \Pr\{A_t = a \mid \mathbf{s}_t = \mathbf{s}, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}.$$

In particular, we consider methods that learn policy parameters on each time step based on the gradient of some scalar performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameters—referred to as *policy gradient methods* (Sutton and Barto, 2018, Chapter 13). Policy gradient methods find a locally optimal solution to the problem of maximizing the objective $J(\boldsymbol{\theta})$ and work by applying gradient ascent, as in the classic variant of policy gradient methods—the REINFORCE algorithm (Williams, 1992):

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}, \\ \nabla J(\boldsymbol{\theta}_t) &\doteq \mathbb{E}\Big[\nabla \ln \big(\pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)\big) G_t\Big]. \end{aligned} \tag{4.1}$$

Some policy gradient methods are a combination of value-based and policy-based methods and are called *actor–critic methods* (Witten, 1977; Barto, Sutton, and Anderson, 1983; Sutton, 1984). The 'actor' refers to the part of the agent responsible for producing the policy, while the 'critic' refers to the part of the agent responsible for producing a value function. In practice, policy-based methods can have high variance when using only an actor; hence, a large number of samples may be needed for the policy to converge. Actor-critic methods solve this problem and aim to reduce this variance by using an actor and a critic together.

Our method is inspired by actor-critic methods. There are two reasons for our focus on actor-critic methods. The first reason is that online policy-based gradient methods are more suitable for incorporating a state-update function compared to implementations of off-policy offline algorithms like deep Q-networks (DQN) (Mnih et al. 2013; 2015). DQN with a state-update function can be resource-inefficient and involve issues of stale updates as batches of stale experience over a long horizon from a replay buffer are used in updates. The second reason is the strong performance of actor-critic-based methods across many domains (Silver et al., 2014; Haarnoja et al., 2018a, 2018b).

We compare our method to the model-free baseline: the actor-critic (AC) method that is based on the difference between estimates at two successive times—known as *1-step Temporal-Difference (TD) learning*. If $\hat{v}(s, \mathbf{w})$ is an estimate of the value of the state, parameterized by $\mathbf{w}$, then using 1-step Temporal-Difference learning, the critic update with a step size $\alpha^{\mathbf{w}}$ is

$$\begin{aligned} \delta_t &\doteq R_{t+1} + \gamma \hat{v}(\mathbf{s}_{t+1}, \mathbf{w}_t) - \hat{v}(\mathbf{s}_t, \mathbf{w}_t), \\ \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha^{\mathbf{w}} \delta_t \nabla \hat{v}(\mathbf{s}_t, \mathbf{w}_t). \end{aligned} \tag{4.2}$$

To update the policy parameters $\boldsymbol{\theta}$ with a step size $\alpha^{\boldsymbol{\theta}}$, instead of using $G_t$—as in REINFORCE (4.1)—we use a policy gradient update with the one-step return:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha^{\boldsymbol{\theta}} \delta_t \frac{\nabla \pi(A_t \mid \mathbf{s}_t, \boldsymbol{\theta}_t)}{\pi(A_t \mid \mathbf{s}_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha^{\boldsymbol{\theta}} \delta_t \nabla \ln \pi(A_t \mid \mathbf{s}_t, \boldsymbol{\theta}_t). \end{aligned} \tag{4.3}$$

## 4.3 Soft-Planner Policy Optimization

In this section, we introduce a soft-planner policy optimization method (SPPO) that includes major components of model-free and model-based RL architectures, which we introduced in Section 2.5. In the parameterized function approximation setting, we denote model-based reinforcement learning components: the model $\mathcal{M}$, the policy or 'actor' $\pi$, and the state-value function or 'critic' $\hat{v}$, and their respective parameters $\boldsymbol{\theta}$, and $\mathbf{w}$. We use the words "interaction" and "episode" interchangeably.

For each time step $t$ of an interaction, we compute the state $\mathbf{s}_t$ from the previous state $\mathbf{s}_{t-1}$, incorporating the information that followed in the transition: the action $A_{t-1}$ and the observation $O_t$ received by the agent after taking action $A_{t-1}$. An agent state $\mathbf{s}_t$ is a representation of the entire interaction's history at time $t$. The state accumulates all the information since the beginning of an interaction: information about the observation, and the previous action and state.

Incorporating a state-update function in the algorithm is necessary to encapsulate the course of dialogue into a compact summary that is useful for choosing future actions (see, mapping model-based reinforcement learning components into voice document editing in Section 3.6). The state $\mathbf{s}_t$ at time $t$ is computed by the *state-update function* $\mathbf{s}_t = u(\mathbf{s}_{t-1}, A_{t-1}, O_t)$. For example, if the user intends to delete five words, but at the first step, the agent deletes only one word, the agent could use the information about this first attempt to delete the remaining four words. The idea behind this is that the state should be sufficient for predicting and controlling the future trajectories and understanding the consequences of the actions, without having to store a complete history.

The SPPO method includes three major parts:

**Part 1**: Planning and acting.

**Part 2**: Updating the policy $\pi$ and the approximate state-value function $\hat{v}$.

**Part 3**: Updating the model weights.

### Part 1. Planning and Acting

Here we describe the first part of the SPPO method. First, we compute a state $\mathbf{s}_t$ using a state update function. Next, for each possible action $a \in \mathcal{A}$, we use the model $\mathcal{M}$ to output the predicted next state $\hat{\mathbf{s}}(\mathbf{s}, a)$ and the predicted reward $\hat{r}(\mathbf{s}, a)$. Then, using the outputs of the model, for each possible action $a \in \mathcal{A}$ we compute a discounted return $\widehat{G}_t$. While we can iterate the model and predict multiple steps ahead at the cost of more computation, we chose one-step iteration to reduce the computational cost. Thus, we use the value estimate of the one-step predicted state—*one-step lookahead*. We chose a one-step lookahead because it is fully online and incremental. If $\gamma \in [0, 1]$ is a parameter for a discount rate (see Section 2.1), then for each action $a \in \mathcal{A}$ the computed return is

$$\widehat{G}_t \doteq \hat{r}(\mathbf{s}_t, a) + \gamma \hat{v}\big(\hat{\mathbf{s}}(\mathbf{s}_t, a), \mathbf{w}_t\big),$$

and we write the vector of returns $\widehat{G}_t^a$ in which each element $i$ is a calculated return for an action $a_i$.

We define a *soft-planner* policy $\pi^{\text{planner}}$ that is computed by applying a softmax

function $\text{softmax}(x_i) \doteq \frac{\exp(x_i)}{\sum_j \exp(x_j)}$ to the estimated returns $\widehat{G}_t^a$:

$$\begin{aligned}\pi^{\text{planner}} &\doteq \text{softmax}(\widehat{G}_t^a) \\ &= \frac{\exp(\widehat{G}_t^a)}{\sum\limits_{a \in \mathcal{A}} \exp(\widehat{G}_t^a)}.\end{aligned}$$

The state-value corresponding to the planner policy is computed as the dot product of the planner policy $\pi^{\text{planner}}$ and the estimated return for each of the actions:

$$v^{\text{planner}}(\mathbf{s}_t) \doteq \widehat{G}_t^a \cdot \pi^{\text{planner}}(a \mid \mathbf{s}_t).$$

**Part 2. Updating Actor and Critic**

In the second part of the SPPO method the agent takes an action and updates the actor and the critic. We make an assumption that the planner policy is a better policy and use it as a reference policy. Using the planner policy $\pi^{\text{planner}}$, the agent takes an action $A_t$: $A_t \sim \pi^{\text{planner}}(\cdot \mid \mathbf{s}_t)$ and receives reward $R_{t+1}$, followed by the observation $O_{t+1}$ (Figure 4.1).

To update the parameters $\mathbf{w}$ of the approximate value function $\hat{v}(\mathbf{s}_t, \mathbf{w}_t)$, we reduce the error by a small amount in the direction of the planner policy $\hat{v}^{\text{planner}}(\mathbf{s}_t)$ using stochastic gradient descent (SGD). Using SGD, we minimize the mean squared error (MSE) between the state-value of the critic $\hat{v}(\mathbf{s}_t, \mathbf{w}_t)$ and the state-value of the planner $\hat{v}^{\text{planner}}(\mathbf{s}_t)$, which is based on the model reward predictions and the critic's estimates

at predicted states. The critic is updated at each time step by the following rule:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\left[\hat{v}^{\text{planner}}(\mathbf{s}_t) - \hat{v}(\mathbf{s}_t, \mathbf{w}_t)\right]^2 \\ &= \mathbf{w}_t + \alpha\left[\hat{v}^{\text{planner}}(\mathbf{s}_t) - \hat{v}(\mathbf{s}_t, \mathbf{w}_t)\right]\nabla\hat{v}(\mathbf{s}_t, \mathbf{w}_t).\end{aligned}$$

To optimize the actor and to encourage exploration, we introduce two modifications to the standard update. As a first modification, we use the relative entropy between the actor policy $\pi$ and the reference planner policy $\pi^{\text{planner}}$, similar to the work of Schulman, Chen, and Abbeel (2017) in which they use a fixed, uniform policy as the reference policy. In the SPPO method we make an assumption that the planner policy is a better policy and instead of a fixed reference policy we use the continually changing planner policy as the reference. Relative entropy is also known as the Kullback-Leibler divergence and is calculated as follows:

$$D_{KL}\Big(\pi^{\text{planner}}||\pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)\Big) \doteq \sum_{a\in\mathcal{A}} \pi^{\text{planner}} \log \frac{\pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)}{\pi^{\text{planner}}(a \mid \mathbf{s}_t)}.$$

As a second modification, we use entropy similar to the work of Mnih et al. (2016) to increase exploration. We denote an entropy term $H$ at time step $t$ that is computed as follows:

$$H_t\big(\pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)\big) \doteq -\sum_{a\in\mathcal{A}} \pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t) \log \pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t).$$

Then the actor parameters are updated as follows:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t - \alpha\nabla\Bigg[\sum_{a\in\mathcal{A}} \pi^{\text{planner}} \log \frac{\pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)}{\pi^{\text{planner}}(a \mid \mathbf{s}_t)} + \beta H_t\Bigg] \\
&= \boldsymbol{\theta}_t - \alpha\nabla \sum_{a\in\mathcal{A}} \pi^{\text{planner}} \log \frac{\pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)}{\pi^{\text{planner}}(a \mid \mathbf{s}_t)} - \beta\nabla\Bigg(-\sum_{a\in\mathcal{A}} \pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t) \log \pi_{\boldsymbol{\theta}}(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)\Bigg) \\
&= \boldsymbol{\theta}_t - \alpha\nabla \sum_{a\in\mathcal{A}} \pi^{\text{planner}} \log \frac{\pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t)}{\pi^{\text{planner}}(a \mid \mathbf{s}_t)} + \beta\nabla \sum_{a\in\mathcal{A}} \pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t) \log \pi(a \mid \mathbf{s}_t, \boldsymbol{\theta}_t).
\end{aligned}$$

**Part 3. Model Update**

In the third part of the SPPO method we update the model weights. First, using the current state $\mathbf{s}_t$, the action taken $A_t$, and the new observation $O_{t+1}$ as an input for the state-update function $u$, we compute the next state $\mathbf{s}_{t+1}$. The model outputs the predicted next state $\hat{\mathbf{s}}(\mathbf{s}_t, a)$ and reward $\hat{r}(\mathbf{s}_t, a)$ using the current state $\mathbf{s}_t$ and action taken $A_t$ as inputs. The objective is to minimize the mean squared error between the model-predicted feature-vectors of the next state and reward and the next state $\mathbf{s}_{t+1}$ computed by the state updated function and the observed reward $R_{t+1}$. The mean squared errors are computed as follows:

$$\begin{aligned}
\text{MSE}_{\text{states}} &\doteq ||\hat{\mathbf{s}}(\mathbf{s}_t, a) - \mathbf{s}_{t+1}||^2, \\
\text{MSE}_{\text{rewards}} &\doteq ||\hat{r}(\mathbf{s}_t, a) - R_{t+1}||^2.
\end{aligned}$$

All three parts are visualized in the Figure 4.1 and described algorithmically below.

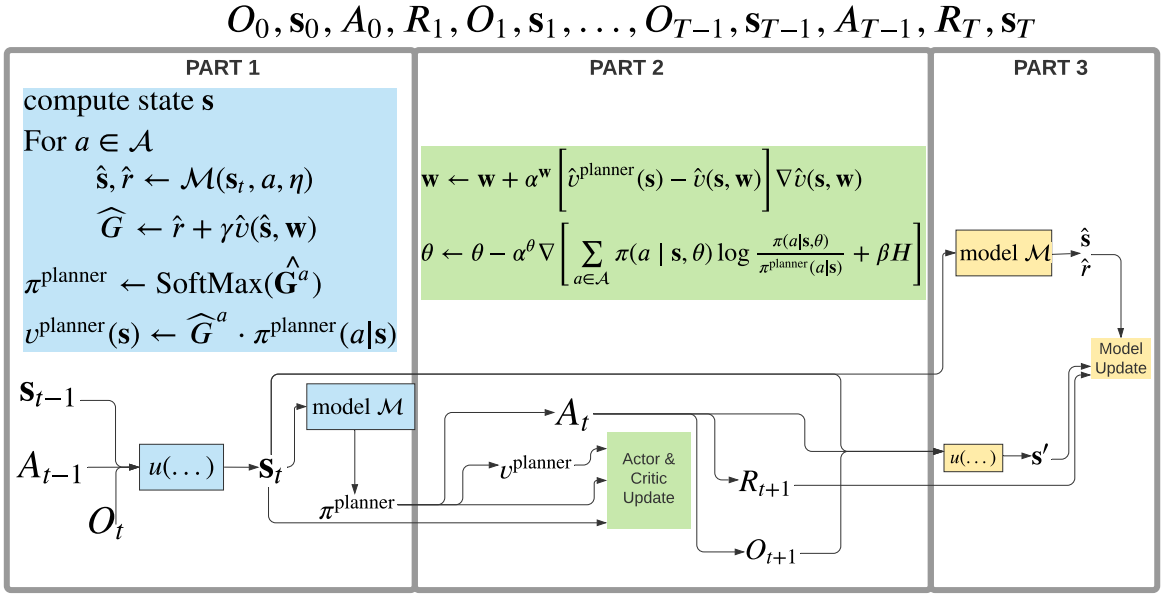


Figure 4.1: SPPO method schematically. The first part is on the left in blue—planning and acting; the second part is in the middle in green—policy and state-value function updates; the third part is on the right in yellow—model update.

**Algorithm:** Soft-Planner Policy Optimization (SPPO)

**Algorithm parameters:** step sizes $\alpha^{\eta}$, $\alpha^{\boldsymbol{\theta}}$, $\alpha^{\mathbf{w}}$; $\gamma, \beta$

Initialize $\mathcal{M}, \pi, \hat{v}, \pi^{\text{planner}}, \boldsymbol{\theta}, \mathbf{w}$, starting state and an action

Observe $O_t$ (first observation of episode)

**while** *still time to train (for each time step t)* **do**

  Compute a state $\mathbf{s}$ by a state-update function

  // Step 1: Plan and act

  **while** *still time to plan* **do**

    **for** $a \in \mathcal{A}$ **do**

      $\hat{\mathbf{s}}(\mathbf{s}, a), \hat{r}(\mathbf{s}, a) \leftarrow \mathcal{M}$ # model outputs next state and reward

      $\widehat{G} \leftarrow \hat{r}(\mathbf{s}, a) + \gamma \hat{v}(\hat{\mathbf{s}}(\mathbf{s}, a), \mathbf{w})$

    **end**

  **end**

  $\pi^{\text{planner}} \leftarrow \text{SoftMax}(\widehat{G}^a)$ # compute a planner policy

  $v^{\text{planner}}(\mathbf{s}) \leftarrow \widehat{G}^a \cdot \pi^{\text{planner}}(a \mid \mathbf{s})$

  Sample action $A \sim \pi^{\text{planner}}(\cdot \mid \mathbf{s})$

  Take action $A$ and get next observation $O$ and reward $R$

  Compute a state $\mathbf{s}$ by a state-update function

  // Update actor $\pi$ and critic $\hat{v}$

  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[ \hat{v}^{\text{planner}}(\mathbf{s}) - \hat{v}(\mathbf{s}, \mathbf{w}) \right] \nabla \hat{v}(\mathbf{s}, \mathbf{w})$ # update critic

  $H \leftarrow -\sum_{a \in \mathcal{A}} \pi(a \mid \mathbf{s}, \boldsymbol{\theta}) \log \pi(a \mid \mathbf{s}, \boldsymbol{\theta})$ # compute entropy

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla \left[ \sum_{a \in \mathcal{A}} \pi^{\text{planner}} \log \frac{\pi(a|\mathbf{s}, \boldsymbol{\theta})}{\pi^{\text{planner}}(a|\mathbf{s})} + \beta H \right]$ # update actor

  Update model

**end**

## 4.4 Experiments

This section presents the experimental setup. First, we investigated the performance of the SPPO method and how quickly the agent learns during planning. Next, we created and investigated the performance of a computationally-light agent. Recall that in the beginning of this chapter we discussed how model-based reinforcement learning methods often result in an agent that is heavier in terms of computation and memory than a model-free agent. One of our goals is that the SPPO method can serve as a solution for a real-life application. With this goal in mind we asked, "Can we first train an agent with the SPPO method, then detach the model and planner components, so that the resulting solution contains only the light model-free architecture that can be deployed as an assistant?" Motivated by this question, we investigated the performance of our agent after we trained the full model-based agent and then detached the model and planner. We called the resulting model-free agent *SPPO-lite*. For evaluation, we used a subset of the dataset: the full dataset of over 74 million sentences was split into halves for training and testing. We used the test set of unseen sentences to assess performance. Instead of sampling actions from a probability distribution, the agent selects actions whose estimated value is the greatest—called *greedy actions* (Sutton and Barto, 2018, Chapter 2).

In all the experiments, each method was executed for $100,000$ simulated interactions, with 30 replications for each method, changing the random seed for each replication. The random seed affects the initialization of parameters of the neural networks. The average over 30 runs was used in the performance measures of the methods. We report average reward per interaction and the distribution over actions. The actions were represented by the discrete action range $[1, 15]$, respecting the real-world setting, where a user is most likely to observe the corrupted words and therefore halt dictation.

### 4.4.1 Word Embeddings

To represent text, we used real-valued vectors that encode the meaning of each word such that the words with similar meanings are closer in the vector space—a technique known as *word embeddings*. We used `word2vec` word embeddings models by Mikolov et al. (2013). We note that the latest developments such as BERT (Devlinet al., 2018) for word embeddings does not apply to our setting. BERT models jointly condition on both the left and right contexts of a sentence, which would create a problem in our simulation provided we inject random noise to simulate corrupted words. Instead, we learn temporal structures with the recurrent neural network state-update function.

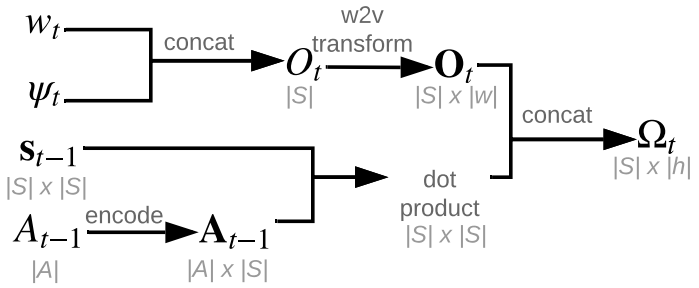


Figure 4.2: Construction of RNN-input $\Omega$ for the state-update function. The $\Omega$ includes the information of the previous state, the observation, and the reward. The dimensions of each array are shown underneath each entity.

### 4.4.2 State Update Architecture

The state-update function $u$, as in (2.4), was represented by a recurrent neural network (RNN): a two-layered bi-directional GRU (Cho et al., 2014) with input and hidden sizes of 400 and 100, respectively. To compute a state using the RNN, we needed to provide an *RNN-input* denoted $\Omega_t$, which combined the previous state, the observation, and the reward (Figure 4.2). The observation $O_t$ was constructed by concatenation of the encoded text $w_t$ and user input $\psi_t$. The `word2vec` embeddings then were used to encode an observation $O_t$ into an *observation matrix* $\boldsymbol{O}_t$. An action was encoded into a binary vector that is all zero values except the index of the integer that represents an action—referred to as *one-hot vectors*. A collection of one-hot vectors results in an *action-matrix* $\boldsymbol{A}_{t-1}$. A dot product of the action-matrix $\boldsymbol{A}_{t-1}$ and the previous state was concatenated with the observation $\boldsymbol{O}_t$ to compute the RNN-input $\Omega$.

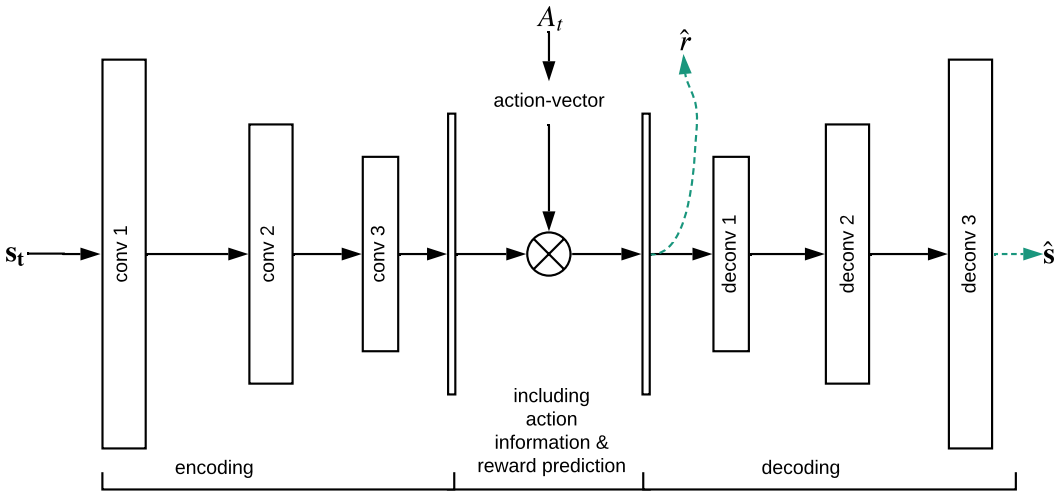


Figure 4.3: A model architecture with a stack of three convolutional layers.

### 4.4.3 Model Architecture

Following Oh et al. (2015), we used a stack of three convolutional layers with filter sizes 50, 50, and 100 for the model architecture (Figure 4.3). The action was encoded into a vector and the Hadamard product was applied to the output of the 1D convolutional layers and the action vector. The output of this operation was then passed through a stack of three deconvolutional layers with filter sizes of 100, 50, and 100. The output of the Hadamard product operation was also passed through a linear layer to a single node for the scalar reward prediction. The state $\mathbf{s}_t$ and the action $A_t$ served as inputs to the model. The model outputs were predictions of the next reward $\hat{r}$ and the next state $\hat{\mathbf{s}}$ similar to other model-based reinforcement learning methods (Kuvayev and Sutton, 1996; Sutton et al., 2008; Hester and Stone, 2012).

### 4.4.4 Baselines Architecture

We use actor-critic as a baseline because actor-critic is often implemented in conjunction with deep learning. We note that often actor-critic is implemented using its asynchronous variants. An example of such a variant is Asynchronous Advantage Actor-Critic (A3C) by Mnih et al. (2016). Asynchronous methods such as A3C are multi-threaded variants that are optimized to simultaneously use multiple CPU threads on a single machine where each thread interacts with its own copy of the environment. In our experiments, we used a single-threaded AC agent that learns from a single stream of experience. A single-threaded agent mimics real human-computer interaction for the task, so the comparison of single-threaded agents is foundational. The actor and critic were implemented as a one-dimensional (1D) convolutional neural network (see Fukushima and Miyake, 1982) that extracts correlations in the temporal structure of the sentences. The network was shared up until the point where there was a split into two separate heads: one producing a scalar value estimate and the other a probability distribution over actions as a policy. The shared architecture consisted of three convolutional layers with filter sizes of 50, 50, and 100.

The supervised learning model implemented the same architecture as the actor in our SPPO method and the training of the supervised learning model was done with cross-entropy loss using the intent as the target.

### 4.4.5 Initialization and Hyperparameters

The architecture was implemented in PyTorch 1.0. To support the required RNN-input—an initial state and an initial action to perform the first state-update iteration—the initial state was initialized to ones, and the initial action was initialized to zeros. We

used the Adam optimizer (Kingma and Ba, 2014) with the default parameters provided by the PyTorch implementation, with the exception of the step size, which was set to $1 \times 10^{-4}$. The step size was selected based on the parameter study performed for each of the methods. Both methods performed the best with the step size of $1 \times 10^{-4}$. We used a discount rate $\gamma$ of 0.9, and the word-embedding size was 300. The entropy parameter $\beta$ was set to 1 for simplicity.

## 4.5 Results

In this section, we present the results on the voice document-editing domain. We compared our method to a model-free actor-critic method that is already known for its good performance across many domains (Schulman et al., 2015, 2017; Haarnoja et al., 2018a, 2018b). As a measure of performance, we used the total sum of reward per interaction, defined as $G_0$ in Section 2.1.1. We averaged $G_0$ over 30 runs, each begun with a different random number seed.

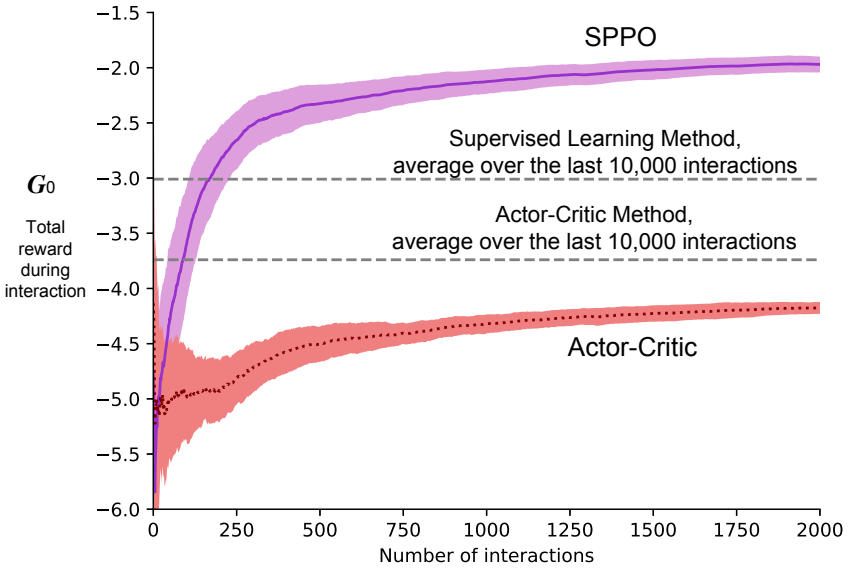


Figure 4.4: Performance of the SPPO method, AC and the supervised learning method on the deletion task. The data are averages over 30 runs of each algorithm, each begun with a different random number seed. Shaded regions are standard error.

### 4.5.1 Sample Efficiency

Figure 4.4 shows the performance of the SPPO method, AC, and the supervised learning method on the deletion task of the voice document-editing domain. The algorithms were run with the selected parameters described in Section 4.4.5. We measured the total reward per interaction, which is the return $G_0$ defined in Section 2.1.1, averaged over 30 runs, each begun with a different random number seed. Each algorithm was executed for 100,000 interactions.

During the first 100 interactions, AC gained a total reward approximately between -5.5 to -4.5, which means that an agent that used AC made 4-5 mistakes on average to correct the corrupted words in a given sentence. AC stabilized its performance

at around 2,000 interactions and improved marginally after that: Figure 4.4 shows that on average, $G_0$ of the last 10,000 interactions for AC was around -3.7, which was not far from its performance at 2,000 interactions. This means that AC made about 3.7 mistakes on average to correct the sentence, which is an improvement from 4-5 mistakes, but not by much.

The supervised learning model gained a higher $G_0$ than AC: it reached an average of -3 over the last 10,000 interactions, compared to -3.7 value for AC. The interpretation of this result is that to correct the sentence about 3 mistakes on average were made by the agent trained with the supervised learning method compared to 3.7 mistakes made by the agent trained with AC method.

The SPPO method gained higher performance much quicker than AC: at around 100 interactions it was approximately at the level of AC's performance at the last 10,000 interactions. At around 250 interactions the SPPO method was already at the level of the supervised learning method's performance and the agent trained with the SPPO method made only about 3 mistakes to correct the sentence. Over the first 2,000 interactions the SPPO method improved its performance about two times, from approximately 4 mistakes to correct the sentence to 2. In comparison, AC did not improve significantly between 100 and 2,000 interactions: only from about 5 to 4 mistakes to correct the sentence. Overall, the SPPO method quickly gained a higher total reward per episode and continued to improve while AC improved slowly and practically plateaued.

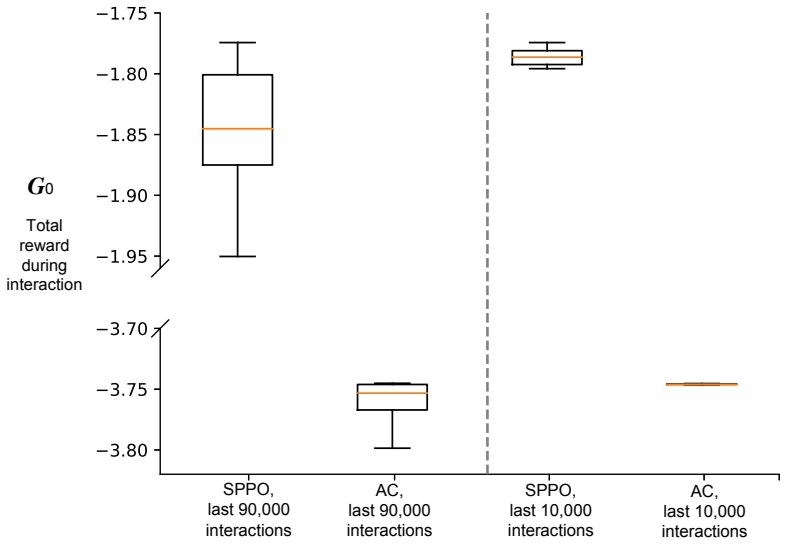


Figure 4.5: The total reward per interaction, the return $G_0$, averaged over all runs. The plot shows the average over the last 90,000 interactions (on the left), and over the last 10,000 interactions (on the right) for the SPPO method and AC. The data are averages over 30 runs of each algorithm, each begun with a different random number seed. The orange line is the median and the extent of the boxes represents upper and lower quartiles.

### 4.5.2 Long-term Performance

Figure 4.5 shows the return $G_0$, over the long period of interactions, averaged over 30 runs, and then further averaged over the last 90,000 interactions (on the left) and the last 10,000 interactions (on the right). Generally, 10,000 interactions between the agent (the assistant) and a user can be a significant amount of interactions. The purpose of averaging over the last large number of interactions was to demonstrate the agent's performance as if the agent continued to assist the user on the selected task in the long run.

First, consider the last 90,000 interactions on the left side of the Figure 4.5. The

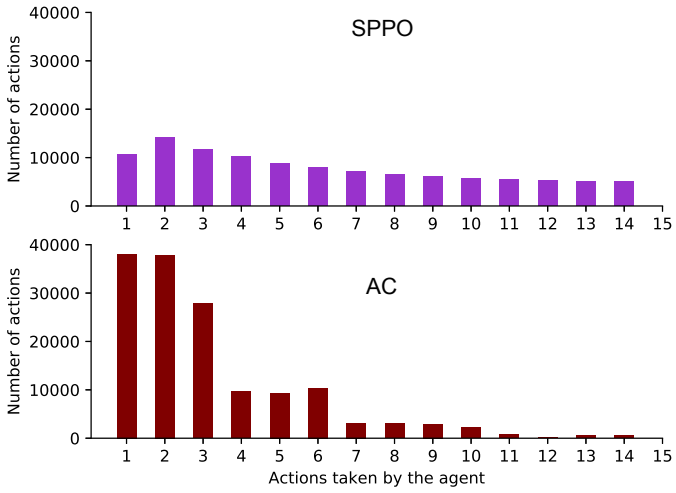


Figure 4.6: The distribution over actions for all actions that the SPPO method and AC took over all interactions.

median for the SPPO method was around $-1.8$. The interquartile interval, where 50% of the data is found, was almost twice as tight for AC than for the SPPO method. The spread between lower and upper quartiles also showed more variability for the SPPO method.

Next, consider the last 10,000 interactions on the right side of the Figure 4.5. Noticeably, the median for the SPPO method was higher over the last 10,000 interactions compared to the last 90,000 interactions, while the median for AC stayed at around the same value. Interestingly, the interquartile interval shortened significantly for both methods. Overall, the SPPO agent made on average 2 fewer mistakes to correct the sentence than the agent trained with AC.

Figure 4.6 shows the distribution of actions selected by the agent for both methods over all interactions. AC preferred lower numbered actions, and seemed to explore less, while SPPO's distribution over actions was much more uniform.

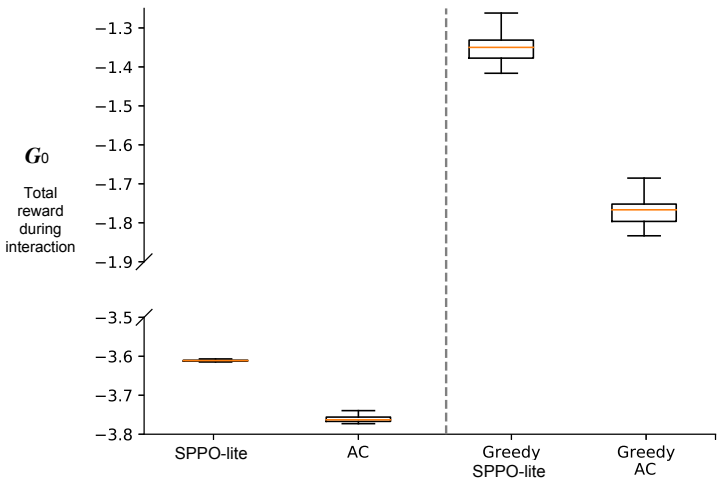


Figure 4.7: The performance of SPPO-lite and AC. SPPO-lite outperformed AC with and without using greedy action selection. The orange line is the median and the extent of the boxes represents upper and lower quartiles.

### 4.5.3 Model-free SPPO-lite Agent

Figure 4.7 shows the performance of the two *model-free architectures*: the SPPO-lite method and AC. We measured the total reward per episode—the return $G_0$. Recall that first we trained a full model-based agent using the SPPO method. Then we detached the model and the planner components so that the resulting solution contained only model-free architecture—the SPPO-lite.

Without greedy action selection (on the left) the SPPO-lite method and AC performed around the same: both methods made close to 3.5 mistakes on average to correct the sentence. Using greedy action selection (on the right) resulted in a different picture: it made just over 1 mistake on average for the SPPO-lite method to correct the sentence while AC made about 2 mistakes.

## 4.6 Discussion

The first result shown in Figure 4.4 suggests that the SPPO method could be more effective than AC and the supervised learning model. It seems that the superior performance of the SPPO method empowered with the model is expected similar to how it would be expected that a supervised learning model would outperform model-free AC methods. However, it is unclear what exactly caused the gain in performance. For example, the SPPO's performance could be explained by improved exploration based on the more uniform distribution over actions. This improved exploration could come from using the model or could come from updates that we proposed for the actor and critic. Recall that the SPPO method is not simply AC with a model, but AC with a model,a planner policy, and a novel update for both actor and critic.

Moreover, these results depend on many choices, for example, choices of parameters, word embeddings, and architectures. This experiment is limited to the selected settings and choices made, and thus we cannot generalize from this result. It is possible that even a different choice of word embeddings would bring the methods closer to each other in performance or could reduced the variability in the SPPO method.

The second result on Figure 4.7 suggests that SPPO-lite could perform about the same as AC, but could be superior to AC using greedy action selection. When selecting the greedy actions, the agent is exploiting its current knowledge. This could be suitable for our goal of deploying a computationally light agent, however, such an assistant would have to be re-trained at some point in time. This experiment demonstrates that we can accomplish our desired goal and distill model and planner knowledge into the model-free architecture. The resulting assistant could be significantly more effective than a regular model-free agent.

Our goal was to demonstrate a sound model-based reinforcement learning method that allows the agent to adapt in the online learning setting, without using built-in knowledge of previously obtained data. The deletion task is difficult for the supervised learning model because it requires learning long-term dependencies in the dictated sentence. Pre-training the agent on large datasets is not a suitable option when it comes to online learning. Learning a model could help the agent to adapt quickly, so it is no surprise that SPPO performs well in the continually changing setting of the deletion task. SPPO learns the language model of the dictated sentences and eventually allows deletion of corrupted words as quickly as two mistakes on average.

## 4.7 Conclusions

In this chapter, we presented a soft-planner policy optimization method that uses a soft-planner policy to update the agent's model-free policy. On the selected parameters and settings, our SPPO method can be more sample-efficient than the classical actor-critic method that does not use the model and the planner. We demonstrated that at deployment time, the model and planner can be detached to obtain an inexpensive, light, yet well-performing model-free agent.

We believe this result and architecture are important in applying RL to real-life applications that use human-computer interactions for a few reasons. The first reason is that this is the first implementation of the voice document-editing domain and the first application of model-based reinforcement learning methods to it. It is a step forward in realizing voice document editing that can directly benefit society by improving productivity and helping those with limited typing abilities. The second reason is that the results demonstrate the difficulty of the deletion task: none of the methods were able to achieve the level of performance necessary to make zero mistakes when correcting sentences. This is realistic and expected for real-life problems. The third reason is that the result demonstrates the complexity of choices when it comes to real-life realization of such assistants and the limitation of conclusions drawn from the choices made.

# Chapter 5

# Planning with Expectation Models for Control

The role of this chapter is to provide the main contribution of this thesis: proving incompatibility of planning with expectation models and linear action-value functions. Further, the chapter builds on the main contribution and presents the fourth and final contribution that is strategies for planning with expectation models.

More specifically, in this chapter we look deeper into planning with expectation models; such models have their own benefits, for example, ease of implementation and scalability. While finding solutions that work for conversational AI and in particular, voice document editing, we also want to find methods that are applicable for building AI assistants of all types. In this chapter, I provide a consistent and systematic approach to planning in model-based reinforcement learning that applies to general settings with function approximation. Value iteration type of planning works by estimating value functions—it is an iterative process that makes value functions better. There are two ways in which value iteration can be done: one way is using *state*-value functions and another way is *action*-value functions. One advantage of action-value functions is that

they immediately tell the agent how to behave. Here, we show that planning with appealing expectation models cannot be done with action-value functions. This brings up the problem of how to do planning with state-value functions and how to effect action selection. We address the problem by considering three strategies for action selection.

## 5.1 Planning with Function Approximation

Methods that resort to approximating functions rather than learning them exactly have not been fully investigated. Extending the techniques used in the simpler *tabular* regime to the *function approximation* regime (see Section 2.3) is an obvious first step, but some of the ideas that have served us well in the past might actually be impeding progress on the new problem of interest. For example, Sutton and Barto (2018) showed that when dealing with feature vectors rather than underlying states, the common Bellman error objective is not learnable with any amount of experiential data. Recently, Naik et al. (2019) showed that discounting is incompatible with function approximation in the case of continuing control tasks. Understanding function approximation in reinforcement learning is key to building general-purpose intelligent assistants that can learn to solve many tasks of arbitrary complexity in the real world.

Planning with function approximation is an interesting and unsolved problem, especially for solving control problems in stochastic environments. Recall that in Section 2.5 we introduced models and planning, and stated that one advantage of models and planning is that they are useful when the agent faces unfamiliar or novel situations—when the agent may have to consider states and actions that it has not experienced or seen before. Planning can help the agent evaluate possible actions by using the model to

compute hypothetical scenarios and then computing their expected future outcomes (Doll et al., 2012; Ha and Schmidhuber, 2018; Sutton and Barto, 2018). Planning with function approximation remains an open question in reinforcement learning today and there are some works in this direction (e.g., Shariff and Szepesvári, 2020).

Planning with function approximation can be performed with various kinds of models: distribution models, sample models, and expectation models. A full distribution model may be impractical and a sample model may be either more expensive computationally or suffer from high variance. Thus, expectation models seem like a good alternative to those choices. However, it is not obvious how expectation models can be used in stochastic environments with function approximation because they only partially characterize a distribution. Wan et al. (2019) considered planning with an expectation model for the prediction problem within the function approximation setting and showed conditions under which the model could produce the expectation of the next feature vector rather than the full distribution, or a sample thereof, with no loss in planning performance. Yet, there is more to be done because Wan et al. considered only planning for prediction to evaluate a fixed policy and not the control problem we are interested in.

In this chapter, we extend Wan et al.'s (2019) work on the prediction problem to the more challenging control problem in the context of stochastic and non-stationary environments. We start off by discussing important choices in model-based reinforcement learning, followed by fundamentals on planning with expectation models in the general context of function approximation. We prove that in stochastic environments planning with an expectation model must update a *state*-value function, not an *action*-value function as previously suggested in the literature (e.g., Sorg and Singh, 2010; van Hasselt et al., 2019; Jafferjee, 2020), by which we open the question of how

planning influences action selection. We consider three ways in which actions can be selected when planning with state-value functions and present general model-based reinforcement learning algorithms for each of them. We demonstrate these algorithms empirically and consider the strengths and weaknesses of these algorithms in computational experiments. Our algorithms are the first to treat model-based reinforcement learning with expectation models in a general setting.

## 5.2 Background

In the previous chapters we discussed that model-based methods are an important part of reinforcement learning's claim to provide a full account of intelligence. An intelligent agent should be able to model its environment and use that model flexibly and efficiently to plan its behavior. In model-based reinforcement learning, models add knowledge to the agent in a way that policies and value functions do not. When planning is applied, the model's output is used to further improve policies and value functions. We now define value iteration with function approximation, and the backup distribution.

### 5.2.1 Value Iteration

A known approach to learning a policy is iterative updates to a state-value or action-value function (Sutton, 1988). Various solutions exist for updating value functions. The value of a state depends on the values of the actions possible in that state and on how likely each action is to be taken under the current policy (Sutton and Barto, 2018, Chapter 3). To iteratively update the estimated value of the state $s$ (or a state–action

pair), usually, the value of its successor states $s'$ (or state–action pairs) is computed first and then the value information is transferred *back* to the state—the process referred to as *backup operations*. The state we are seeking to update is called the *backup state*.

*Value iteration* is a classic planning algorithm (Bellman, 1957; Sutton, 1988; Watkins, 1989; Bertsekas, 2012) that performs sweeps of computed *backup values* through the state or state-action space. Using value functions, value iteration for all $s \in \mathcal{S}$ can be written as follows for the tabular case:

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_k(s')\big], \quad s \in \mathcal{S}. \end{aligned}$$

Similarly, we can write the corresponding value iteration for the function approximation case. Recall that, in the function approximation setting, the agent state is represented by a feature vector $\mathbf{s} \in \mathbb{R}^d$, that is computed using a *state-update* function $u$ (Sutton and Barto, 2018) that uses the most recent observation and action along with the most recent agent state to recursively compute the new agent state as in (2.4). Consider a transition probability function $p(\mathbf{s}' \mid \mathbf{s}, a)$ that takes the agent from state $\mathbf{s}$ to a successor state $\mathbf{s}'$, $p : \mathbb{R}^d \times \mathbb{R}^d \times \mathcal{A} \to [0, 1]$, such that

$$\int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}' \mid \mathbf{s}, a) d\mathbf{s}' \leq 1, \quad \mathbf{s} \in \mathbb{R}^d, \quad a \in \mathcal{A}.$$

Consider an approximate value function $\hat{v}(\mathbf{s}, \mathbf{w}) \approx v_*$ and its weight vector $\mathbf{w}$. For this update we select the states in some order and for each single state we change the weight vector $\mathbf{w}$ on each update. We refer to such an update as an *iteration* $k$ and we include the iteration number as a superscript not to confuse it with the actual weight

vector and state at time $t$ which is a separate sequence. The difference is the number of iterations corresponds to the *number of planning steps* and we are not necessarily doing one step of planning in parallel to every step in the real world that is denoted by time steps $t$. The choice of the number of iterations—number of planning steps—is an algorithmic choice, while $\mathbf{s}_t$ sequence is based on the evolution of the environment.

In this update notation we drop the time index $t$ and we consider an update for each set of states. Then *Approximate Value Iteration (AVI)* consists of repeated applications of the following update for a given state $\mathbf{s}$ and a weight $\mathbf{w}^k \in \mathbb{R}^d$ at the iteration $k$ with a positive step-size parameter $\alpha$, and the gradient vector $\nabla$ with respect to $\mathbf{w}^k$ is as follows:

$$\mathbf{w}^{k+1} \doteq \mathbf{w}^k + \alpha \left( \max_a \left[ r(\mathbf{s}^k, a) + \gamma \int_{\mathbf{s}' \in \mathbb{R}^d} p\big(\mathbf{s}' \mid \mathbf{s}, a\big) d\mathbf{s}' \hat{v}\big(\mathbf{s}', \mathbf{w}^k\big) \right] - \hat{v}(\mathbf{s}, \mathbf{w}^k) \right) \nabla \hat{v}(\mathbf{s}, \mathbf{w}^k). \tag{5.1}$$

### 5.2.2 Planning and Backup Distribution

We discussed above that planning can often be described as proceeding in a sequence of *state-value backups*, each of which updates the value estimate of a single state (and perhaps others by generalization). The backup state can be selected in many different ways. Often we can speak of the distribution of states at which backups are done. A state distribution from which backup updates are performed during planning has an effect on the value function convergence. Here, we refer to the distribution of states we are planning from as a *backup distribution*. The process of choosing this distribution is known as *backup control* which is analogous to "search control" strategies such as prioritized sweeping (see Sutton et al., 2008; Pan et al., 2018). To update the backup

state, one or more backups are made either from the backup state or from its possible successor states, a backup target is formed, and finally the backup-state's state-value estimate is shifted toward the target. In one-step backups, all the backup updates are from the same backup state; they all predict one step into the future from it. These are the shortest backups. In *multi-steps*, or *iterated backups*, backups are made both from the backup state and the predicted states of those backups.

## 5.3 Equivalence of Planning with Expectation and Distribution Models

In this section, we show that planning with expectation models is equivalent to planning with distribution models when the value function is linear. To do so, we first introduce a few important definitions.

**Definition 5.3.1** (Distribution Model)**.** A *distribution model* consists of an *expected reward function* $r : \mathbb{R}^d \times \mathcal{A} \to \mathbb{R}$, written $r(\mathbf{s}, a)$, and a *transition probability function* $p : \mathbb{R}^d \times \mathbb{R}^d \times \mathcal{A} \to [0, 1]$, written $p(\mathbf{s}' \mid \mathbf{s}, a)$, such that

$$\int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}' \mid \mathbf{s}, a) d\mathbf{s}' \leq 1, \quad \mathbf{s} \in \mathbb{R}^d, \quad a \in \mathcal{A}.$$

If $p$ integrates to less than 1, then the remaining probability is interpreted as the probability of termination.

**Definition 5.3.2** (AVI with a distribution model)**.** Following an approximate update for a weight vector in (5.1) we define an *update target* $g(\mathbf{s}, \mathbf{w})$ for a large (or infinite)

state space as follows:

$$g(\mathbf{s}, \mathbf{w}) \doteq \max_{a \in \mathcal{A}} \left[ r(\mathbf{s}, a) + \gamma \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}' \mid \mathbf{s}, a) \hat{v}(\mathbf{s}', \mathbf{w}) d\mathbf{s}' \right]. \tag{5.2}$$

Then *AVI with a distribution model* consists of repeated applications of the following update for a given state $\mathbf{s}$ and a weight $\mathbf{w}^k \in \mathbb{R}^d$ at the iteration $k$ with a positive step-size parameter $\alpha$, and the gradient vector $\nabla$ with respect to $\mathbf{w}^k$ is

$$\mathbf{w}^{k+1} \doteq \mathbf{w}^k + \alpha \Big( g(\mathbf{s}, \mathbf{w}) - \hat{v}(\mathbf{s}, \mathbf{w}^k) \Big) \nabla \hat{v}(\mathbf{s}, \mathbf{w}^k). \tag{5.3}$$

**Definition 5.3.3** (GEEM)**.** A *general episodic expectation model (GEEM)* consists of an expected reward function $r : \mathbb{R} \times \mathcal{A} \to [-\infty, 0]$, an expected next-state function $\hat{\mathbf{s}} : \mathbb{R}^d \times \mathcal{A} \to \mathbb{R}^d$, and a termination probability function $\beta : \mathbb{R}^d \times \mathcal{A} \to [0, 1]$. The termination probability function $\beta(\mathbf{s}, a)$ is interpreted as the probability of terminating in one step from state $s$ if action $a$ is taken, and $\hat{\mathbf{s}}(\mathbf{s}, a)$ is interpreted as the expected next state if termination does not occur.

**Definition 5.3.4** (EVI)**.** *Expectation value iteration (EVI)* consists of repeated application of AVI (5.3) with the update target

$$g(\mathbf{s}, \mathbf{w}) \doteq \max_{a \in \mathcal{A}} \left[ r(\mathbf{s}, a) + \gamma \big(1 - \beta(\mathbf{s}, a)\big) \hat{v}\big(\hat{\mathbf{s}}(\mathbf{s}, a), \mathbf{w}\big) \right]. \tag{5.4}$$

In the special case in which the approximate value function is linear, $\hat{v}(\mathbf{s}, \mathbf{w}) = \mathbf{w}^\top \mathbf{s}$, this update target can be written

$$g(\mathbf{s}, \mathbf{w}) = \max_{a \in \mathcal{A}} \left[ r(\mathbf{s}, a) + \gamma \underbrace{\big(1 - \beta(\mathbf{s}, a)\big) \hat{\mathbf{s}}(\mathbf{s}, a)}_{\bar{\mathbf{s}}(\mathbf{s}, a)}{}^\top \mathbf{w} \big) \right], \tag{5.5}$$

revealing that the $\beta$ and $\hat{\mathbf{s}}$ functions of the expectation model can be combined and learned as a single function $\bar{\mathbf{s}}(\mathbf{s}, a)$. This motivates the next definition.

**Definition 5.3.5** (ZTEM)**.** A *zero-terminal expectation model (ZTEM)* consists of an expected reward function $r : \mathbb{R} \times \mathcal{A} \to [-\infty, 0]$ and a zero-terminal expected next-state function $\bar{\mathbf{s}} : \mathbb{R}^d \times \mathcal{A} \to \mathbb{R}^d$, written $\bar{\mathbf{s}}(\mathbf{s}, a)$, which can be interpreted as an expectation of the next state given that the terminal state is treated as a zero vector.

**Definition 5.3.6** (ZTEM and a GEEM are *aligned*)**.** A ZTEM and a GEEM are *aligned* iff their expected reward functions are identical and the expected next state functions are related by

$$\bar{\mathbf{s}}(\mathbf{s}, a) = \big(1 - \beta(\mathbf{s}, a)\big)\hat{\mathbf{s}}(\mathbf{s}, a). \tag{5.6}$$

**Definition 5.3.7** (LEVI)**.** *Linear expectation value iteration (LEVI)* is the combination of EVI and a linear state-value function. It consists of repeated applications of (5.3) with the update target

$$g(\mathbf{s}, \mathbf{w}) \doteq \max_{a \in \mathcal{A}} \big[r(\mathbf{s}, a) + \gamma \mathbf{w}^\top \bar{\mathbf{s}}(\mathbf{s}, a)\big]. \tag{5.7}$$

**Theorem 1.** If a ZTEM and GEEM are aligned, then LEVI is equivalent to EVI with a linear state-value function.

*Proof.* Follows immediately from the combination of (5.5) and (5.6). In particular, in (5.6) we stated the relation between the expected next state functions:

$$\bar{\mathbf{s}}(\mathbf{s}, a) = \big(1 - \beta(\mathbf{s}, a)\big)\hat{\mathbf{s}}(\mathbf{s}, a).$$

Expectation value iteration from (5.5) is as follows:

$$\begin{aligned} g(\mathbf{s}, \mathbf{w}) &= \max_{a \in \mathcal{A}} \Big[ r(\mathbf{s}, a) + \gamma \underbrace{\big(1 - \beta(\mathbf{s}, a)\big)\hat{\mathbf{s}}(\mathbf{s}, a)}_{\bar{\mathbf{s}}(\mathbf{s}, a)}{}^{\top} \mathbf{w}\big) \Big] \\ &= \max_{a \in \mathcal{A}} \big[ r(\mathbf{s}, a) + \gamma \bar{\mathbf{s}}(\mathbf{s}, a)^{\top} \mathbf{w} \big] \\ &= \max_{a \in \mathcal{A}} \big[ r(\mathbf{s}, a) + \gamma \mathbf{w}^{\top} \bar{\mathbf{s}}(\mathbf{s}, a) \big]. \end{aligned}$$

This shows that the update target in LEVI (5.7) is the same as the update target in EVI (5.4), thus, LEVI is equivalent to EVI.

□

**Definition 5.3.8** (ZTEM and a distribution model are aligned)**.** A *ZTEM and a distribution model are aligned* iff their expected reward functions are identical and their transition parts are related by:

$$\bar{\mathbf{s}}(\mathbf{s}, a) = \int_{\mathbf{s}' \in \mathbb{R}^d} \mathbf{s}' p(\mathbf{s}' \mid \mathbf{s}, a) d\mathbf{s}' \quad \forall\ \mathbf{s} \in \mathbb{R}^d, a \in \mathcal{A}. \tag{5.8}$$

The theorem below extends Wan et al.'s (2019) result to general setting of function approximation.

**Theorem 2.** If a ZTEM and a distribution model are aligned, then LEVI is equivalent to AVI with a linear state-value function.

*Proof.* Consider now the integral part in (5.2) below:

$$\begin{aligned}
\int_{\mathbf{s}'\in\mathbb{R}^d} p(\mathbf{s}'|\mathbf{s},a)\hat{v}(\mathbf{s}',\mathbf{w})d\mathbf{s}' &= \int_{\mathbf{s}'\in\mathbb{R}^d} p(\mathbf{s}' \mid \mathbf{s},a)\mathbf{w}^\top \mathbf{s}' d\mathbf{s}' \\
&= \int_{\mathbf{s}'\in\mathbb{R}^d} \mathbf{w}^\top p(\mathbf{s}' \mid \mathbf{s},a)\mathbf{s}' d\mathbf{s}' \\
&= \mathbf{w}^\top \int_{\mathbf{s}'\in\mathbb{R}^d} p(\mathbf{s}' \mid \mathbf{s},a)\mathbf{s}' d\mathbf{s}' \\
&= \mathbf{w}^\top \mathbb{E}_{\mathcal{F},\boldsymbol{\eta}}\big[\mathbf{s}_{t+1} \mid \mathbf{s}_t = \mathbf{s}, A_t = a\big] \\
&= \mathbf{w}^\top \bar{\mathbf{s}}(\mathbf{s},a) \\
&\doteq \bar{\mathbf{s}}(\mathbf{s},a)^\top \mathbf{w}.
\end{aligned}$$

This shows that AVI and LEVI are equivalent.

$$\begin{aligned}
g(\mathbf{s},\mathbf{w}) &= \max_{a\in\mathcal{A}} \left[ r(\mathbf{s},a) + \gamma \int_{\mathbf{s}'\in\mathbb{R}^d} p(\mathbf{s}' \mid \mathbf{s},a)\hat{v}(\mathbf{s}',\mathbf{w})d\mathbf{s}' \right] \text{ (AVI)} \\
&= \max_{a\in\mathcal{A}} \big[ r(\mathbf{s},a) + \gamma \mathbf{w}^\top \bar{\mathbf{s}}(\mathbf{s},a) \big], \text{ (LEVI).}
\end{aligned}$$

□

The theorem allows us to use the benefits of expectation models such as lesser computation and smaller variance.

## 5.4 Incompatibility of Expectation Models and Action-Value Functions

In this section we show that in stochastic environments planning with expectation models, function approximation, and linear value functions must update state-value functions and not action-value functions as previously suggested in the literature (e.g., by Sorg and Singh, 2010; van Hasselt et al., 2019; Jafferjee, 2020).

It is natural to extend approximate value iteration to an action-value form. An *update target* for an *approximate action-value iteration* (AAVI) for a given state $\mathbf{s}$ and a weight $\mathbf{w}^k \in \mathbb{R}^d$ at the iteration $k$ with a positive step-size parameter $\alpha$, and the gradient vector $\nabla$ with respect to $\mathbf{w}^k$ is defined as:

$$g(\mathbf{s}, a, \mathbf{w}) \doteq r(\mathbf{s}, a) + \gamma \int\limits_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}' \mid \mathbf{s}, a) \max_{a'} \hat{q}(\mathbf{s}', a', \mathbf{w}) d\mathbf{s}'. \tag{5.9}$$

The AAVI update is then

$$\mathbf{w}^{k+1} \doteq \mathbf{w}^k + \alpha\big(g(\mathbf{s}, a, \mathbf{w}^k) - \hat{q}(\mathbf{s}, a, \mathbf{w}^k)\big)\nabla \hat{q}(\mathbf{s}, a, \mathbf{w}^k). \tag{5.10}$$

Based on our Theorem 2, we can write the integral part in the update target for AAVI in (5.9) as follows:

$$\int\limits_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \max_{a'} \hat{q}(\mathbf{s}', a')\, d\mathbf{s}' \doteq \int\limits_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \max_{a'} (\mathbf{w}_{a'}^\top \mathbf{s}')\, d\mathbf{s}'. \tag{5.11}$$

However, we also know that

$$\int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \max_{a'} (\mathbf{w}_{a'}^\top \mathbf{s}') \, d\mathbf{s}' \neq \max_{a'} \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \mathbf{w}_{a'}^\top \mathbf{s}' d\mathbf{s}'. \tag{5.12}$$

Combining (5.11) and (5.12) we get

$$\begin{aligned} \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \max_{a'} \hat{q}(\mathbf{s}', a') \, d\mathbf{s}' &\doteq \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \max_{a'} (\mathbf{w}_{a'}^\top \mathbf{s}') \, d\mathbf{s}' \\ &\neq \max_{a'} \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \mathbf{w}_{a'}^\top \mathbf{s}' d\mathbf{s}'. \end{aligned} \tag{5.13}$$

We now substitute the update target (5.9) in the AAVI update (5.10) with (5.13) and replace the part of the integral with the output of the ZTEM as per Theorem 2:

$$\begin{aligned} \mathbf{w}^{k+1} &= \mathbf{w}^k + \alpha \left[ r(\mathbf{s}, a) + \gamma \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \max_{a'} \hat{q}(\mathbf{s}', a') \, d\mathbf{s}' - \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \right] \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \\ &= \mathbf{w}^k + \alpha \left[ r(\mathbf{s}, a) + \gamma \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \max_{a'} (\mathbf{w}_{a'}^\top \mathbf{s}') \, d\mathbf{s}' - \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \right] \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \\ &\neq \mathbf{w}^k + \alpha \left[ r(\mathbf{s}, a) + \gamma \max_{a'} \int_{\mathbf{s}' \in \mathbb{R}^d} p(\mathbf{s}'|\mathbf{s}, a) \mathbf{w}_{a'}^\top \mathbf{s}' d\mathbf{s} - \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \right] \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \\ &\neq \mathbf{w}^k + \alpha \left[ r(\mathbf{s}, a) + \gamma \max_{a'} \left( \bar{\mathbf{s}}(\mathbf{s}, a)^\top \mathbf{w} \right) - \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \right] \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}^k) \text{ (by Theorem 2).} \end{aligned}$$

Thus, the backup value obtained from the distribution model is not aligned with the backup value obtained from the expectation model. Hence, we state an important result: in stochastic environments, planning with expectation models and linear value functions cannot proceed with action-value functions. We refer to this result as the *incompatibility of expectation models and action-value functions.*

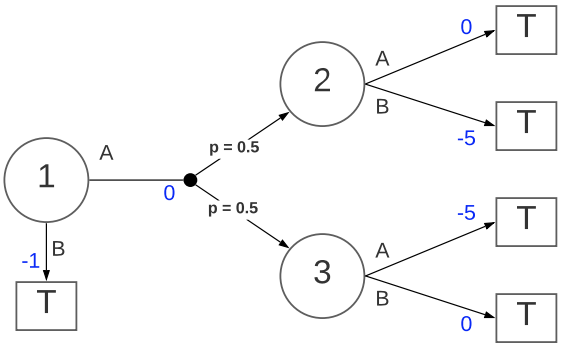


Figure 5.1: A counterexample episodic MDP.

### 5.4.1 A Counterexample Illustration

We now empirically demonstrate the incompatibility of expectation models and action-value functions on a counterexample. Consider a Markov decision process with three transition states shown in Figure 5.1. From the start state 1 the agent can perform one of two actions $A$ and $B$. With equal probability, the action $A$ causes the agent to advance either to state 2 or 3. If the transition takes the agent to state 2, then action $A$ takes the agent to the terminal state with the reward 0, and action $B$ takes the agent to the terminal state with the reward $-5$. The opposite happens in state 3: action $A$ takes the agent to the terminal state with the reward $-5$, and action $B$ takes the agent to the terminal state with the reward 0. From state 1, the action $B$ takes the agent to the terminal state with the reward $-1$. The episode ends once the agent reaches the terminal state.

In this experiment, we compared the model-free Q-learning method (see Section 2.4) and the AAVI method that used a learned expectation model, as in (5.10). We used $\epsilon$-greedy action selection with $\epsilon = 0.1$, meaning that the agent takes actions greedily

most of the time—maximizing the reward according to the $\arg\max_a$ function—but every once in a while with a small probability $\epsilon$, instead selects randomly among all the actions with equal probability, independently of the action-value estimates. We measured a sum of all the rewards since the beginning of the episode—the return $G_0$. The data were averages over 100 runs, each begun with a different random number seed.

With greedy action selection, the optimal policy for Q-learning would get the return $G_0 = 0$. However, since we used $\epsilon$-greedy, then the agent picked an optimal action with probability $p = 1 - \epsilon + \frac{\epsilon}{2} = 0.95$. Then the probability of picking the optimal action at any state was $(1-p)(-1) + p(1-p)(-5) = 0.05 \cdot (-1) + 0.95 \cdot 0.05 \cdot (-5) = -0.29$. This means that the maximum return that Q-learning could receive is around -0.3. Similarly, we expect AAVI with the expectation model to obtain a performance of -1 using greedy action selection, and -0.7 using $\epsilon$-greedy.

We used tabular features meaning that states were encoded as *one-hot vectors*—binary vectors that are all zero values except the index of the integer that represents a state. We used 0.3 for the action-values step size in both Q-learning and AAVI, and 0.3 for the expectation model step-size. We set the number of planning steps $k = 20$.

The model was updated as per below. The model consists of a forward transition matrix $\mathcal{F} \in \mathbb{R}^d \times \mathbb{R}^d$ and an expected reward vector $\boldsymbol{\eta} \in \mathbb{R}^d$ constructed such that $\hat{\mathbf{s}}(\mathbf{s}, a, \boldsymbol{\eta}) \doteq \mathcal{F}\mathbf{s}$ and $\hat{r}(\mathbf{s}, a, \boldsymbol{\eta}) \doteq \boldsymbol{\eta}^\top \mathbf{s}$ can be used as estimates of the feature vector and reward that follow the state $\mathbf{s}_t$. Then the model update with step size $\alpha^{\mathcal{F}}$ is as follows (see Appendix A for details):

$$\begin{aligned}
\mathcal{F}_{t+1} &\doteq \mathcal{F}_t + \alpha^{\mathcal{F}} \Big[ \mathcal{F}\mathbf{s}_t - \mathbf{s}_{t+1} \Big] \mathbf{s}_t^\top, \\
\boldsymbol{\eta}_{t+1} &\doteq \boldsymbol{\eta}_t + \alpha^{\mathcal{F}} [r - \boldsymbol{\eta}_t^\top \mathbf{s}_t] \mathbf{s}_t.
\end{aligned} \tag{5.14}$$

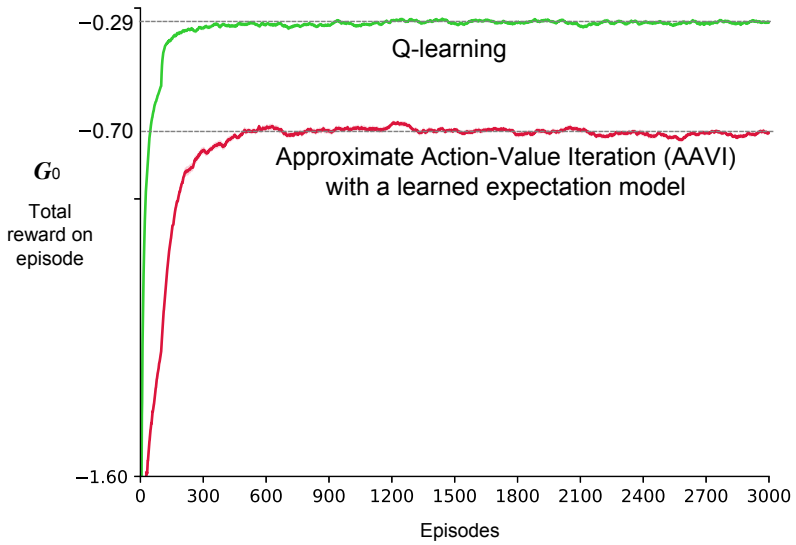


Figure 5.2: Performance of Q-learning and AAVI with a learned expectation model with $\epsilon$-greedy action selection on the counterexample episodic MDP. The data were averaged over 100 runs, each begun with a different random number seed. Shaded regions (barely visible) are standard error.

Figure 5.2 shows the performance of Q-learning and AAVI with an expectation model with $\epsilon$-greedy action selection. Q-learning reached the optimal policy of approximately $-0.3$, while AAVI with a learned expectation model only reached approximately -0.7. This means that AAVI learned to take action $B$ from state 1 while Q-learning learned an optimal policy of taking action $A$ from state 1.

The learned expectation model cannot differentiate between states 2 and 3 because as an expected state they look exactly the same, and as a result AAVI with an expectation model learns a suboptimal policy.


Figure 5.3: A stochastic corridor. The actions are stochastic: with probability $p$, the actions cause the agent to move one cell in the direction corresponding to its name (left or right), and with probability $p-1$, the agent ends up one cell in the other direction. The reward is $-1$ on all time steps except the transitions that take the agent to terminal states. If the agent reaches the terminal state marked by 'G'—the goal state—it gets a reward of $+20$; the reward is 0 for reaching the other terminal state. The episode ends once the agent reaches a terminal state.

### 5.4.2 A Stochastic Corridor Illustration

To illustrate further the incompatibility of expectation models and action-value functions, we compared three methods: the model-free Q-learning method (see Section 2.4), the AAVI method, as in (5.10), that uses a learned expectation model, and the AAVI method that uses the true model of the environment. Consider a stochastic corridor shown on Figure 5.3. The corridor consists of 9 white non-terminal states. The black cells represent walls, red cells represent terminal states. An episode starts with the agent randomly initialized in one of the white states numbered 2–10, and ends when it reaches one of two terminal states, numbered 1 and 11. The agent can move left or right using the `left` or `right` actions. The actions are stochastic: with probability $p$, the actions cause the agent to move one cell in the direction corresponding to its name (left or right), and with probability $p-1$, the agent ends up one cell in the other direction. We refer to such a setup as a *stochasticity* of the environment with a value $\zeta = p - 1$. The reward is $-1$ on all time steps except the transitions that take the agent to terminal states. If the agent reaches the terminal state marked by 'G'—the

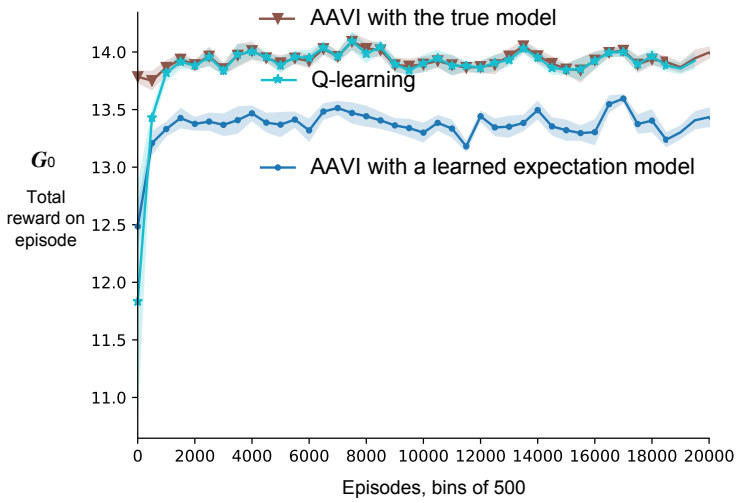


Figure 5.4: Performance of Q-learning and AAVI with expectation model on the corridor stochastic environment with function approximation. Feature vectors of size $d = 14$ are random binary feature vectors for each state. They all have the same $k$ number of 1s, $k = 5$, picked at random, without replacement. Each data point represents the average number of total reward per episode, averaged over all the runs and over temporal stretches of 500-episode bins. Shaded regions are standard error. Stochasticity of the environment was $\zeta = 0.1$.

goal state—it gets a reward of +20; the reward is 0 for reaching the other terminal state.

In this experiment, we used a feature vector of size $d = 14$ with 5 random bits, meaning that it was a vector of all zeros and 5 ones at random. We used $\epsilon$-greedy action selection during decision time, with $\epsilon = 0.1$. We used 0.01 for the action-values step size in both methods, the Q-learning method and the AAVI method, and 0.1 for the expectation model step-size. We set the number of planning steps $k = 20$ and the stochasticity of the environment was $\zeta = 0.1$. The model was updated as in (5.14).

Figure 5.4 shows the performance of Q-learning, AAVI that uses a learned expectation model, and AAVI with the true model, over 20,000 episodes. We measured the

sum of all the rewards since the beginning of the episode—the return $G_0$. The data were averaged over 30 runs, each begun with a different random number seed. Each data point represents the total sum of rewards per episode—$G_0$, averaged over all the runs and over temporal stretches of 500-episode bins where we used binning of episodes to help read the results.

Q-learning started at around 11 and learned the optimal policy within the first 2,000 episodes. The optimal policy is close to 14.5 which we can calculate by solving a system of linear equations for all the state values. AAVI with the true model started much higher right away: its performance was over 13.5 in the beginning which is expected because the true model would always correctly calculate the next state. As a result of planning with the true model, the agent could improve its value function more quickly. AAVI with the learned expectation model only reached about 13.5 which is far from the optimal policy of 14.

This example demonstrates the incompatibility of expectation models and action-value functions on the stochastic corridor.

### 5.4.3 Discussion and Conclusion

Here, we demonstrated that in stochastic environments, planning with expectation models with function approximation and linear value functions is incompatible with *action-value* functions. Our results show that Sorg and Singh's (2010) usage of an expectation model for planning with linear action-value function can be invalid. Similarly, van Hasselt et al. (2019) showed the divergence of forward Dyna in stochastic environments, advocating to use replay-based models. In their work, they explained the divergence due to the the fact that the expected state may not be a valid state. However, we argue that the expected state may be be still a reasonable generalization of a state, especially in the environments with rich features and we showed that the divergence occurs due to the incompatibility of expectation models and action-value functions. Another example is the work of Jafferjee (2020) who hypothesised that AAVI with a learned expectation model may fail to learn control policies and argued this occurs because the imperfect model may erroneously generate fictitious states that do not correspond to real states. Their argument remains correct, but it does not have specifics that we provided here of why the failure occurs. Finally, there are some works (e.g., Yao et al., 2009) that applied AAVI with expectation models only to deterministic environments and thus the problem did not surface for them.

This incompatibility is not restricted to function approximation settings. In practice, the counterexample in Figure 5.1 is implemented with tabular settings extending the incompatibility of planning with expectation models and action-value functions beyond function approximation. Importantly, we can say that the result is not restricted to linear value functions because our counterexample uses tabular value functions which are not linear.

Finally, the incompatibility of expectation models and action-value functions in stochastic environments leads us to an important question that we discuss in the next section.

## 5.5 Action Selection Strategies

The incompatibility of expectation models and action-value functions that we stated in Section 5.4 raises an important question: *how can we affect action selection when planning* if we cannot plan with action-value functions and select actions based on their estimated action-values? In this section, we address this problem and consider three ways of selecting actions when planning with expectation models and state-value functions. To refer to these ways we use the phrase *action selection strategies* to emphasize this important concept of the thesis. There might be other action selection strategies that exist in the literature. We consider the most obvious ones and summarize them into the algorithms at the end of this section. Our objective is to present some theoretically-grounded general ways in which action selection can be done.

### 5.5.1 Definitions

Here, we discuss a few concepts and definitions that are used across all three action selection strategies. We use the concept of *decision time* to discuss how the agent selects actions during the decision time when it needs to take an action and how we update the policy and value functions after that. We use the concept of *planning time* to discuss how we update the policy and value functions during planning.

Recall that $\mathbf{s}$ is a feature vector of an agent state, $\mathbf{s} \in \mathbb{R}^d$. $\bar{\mathbf{s}}(\mathbf{s}, a)$ is an expected next state function and an output of the ZTEM that is aligned with GEEM, defined in (5.6). Also, recall that the target for the regular TD update (see Section 4.2) is $R_{t+1} + \hat{v}(\mathbf{s}_{t+1}, \mathbf{w}_t)$.

Replacing the next state $\mathbf{s}_{t+1}$ and the reward by the outputs of the ZTEM model we define a *ZTEM TD target*:

$$\begin{aligned} b(\mathbf{s}_t, a, \mathbf{w}_t) &\doteq r(\mathbf{s}_t, a) + \gamma \hat{v}\big(\bar{\mathbf{s}}(\mathbf{s}_t, a), \mathbf{w}_t \\ &= r(\mathbf{s}_t, a) + \gamma \bar{\mathbf{s}}(\mathbf{s}_t, a)^\top \mathbf{w}_t. \end{aligned} \tag{5.15}$$

We define a *regular planning update* that consists of three steps. The first step is to sample a state $\mathbf{s}$ from the backup distribution. The second step is to compute the update target by acting greedily with respect to the model based on the values of ZTEM TD target:

$$g(\mathbf{s}_t, \mathbf{w}_t) \doteq \max_{a \in \mathcal{A}} \big[ r(\mathbf{s}_t, a) + \gamma \hat{v}\big(\bar{\mathbf{s}}(\mathbf{s}_t, a), \mathbf{w}_t \big] \tag{5.16}$$

$$= \max_{a \in \mathcal{A}} \big[ r(\mathbf{s}_t, a) + \gamma \bar{\mathbf{s}}(\mathbf{s}_t, a)^\top \mathbf{w}_t \big] \tag{5.17}$$

$$= \max_{a \in \mathcal{A}} \big[ r(\mathbf{s}_t, a) + \gamma \mathbf{w}_t^\top \bar{\mathbf{s}}(\mathbf{s}_t, a) \big] \tag{5.18}$$

$$= \max_{a \in \mathcal{A}} b(\mathbf{s}_t, a, \mathbf{w}_t). \tag{5.19}$$

The third step is to update the parameters of the approximate state-value function:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \bigg( g(\mathbf{s}_t, \mathbf{w}_t) - \mathbf{w}_t^\top \mathbf{s}_t \bigg) \nabla (\mathbf{w}_t^\top \mathbf{s}_t). \tag{5.20}$$

### 5.5.2 Strategy 1: Decide-Time Planning

Here we discuss the first strategy for action selection when planning with the state-value function. In this strategy, during decision time, the agent uses the ZTEM TD target (5.15) as a one-step lookahead by the model and then acts greedily:

$$A_t \doteq \arg\max_{a \in \mathcal{A}} b(\mathbf{s}_t, a, \mathbf{w}_t), \tag{5.21}$$

and then the agent uses a regular TD update:

$$\begin{aligned} \delta_t &\doteq R_{t+1} + \gamma \hat{v}(\mathbf{s}_{t+1}, \mathbf{w}_t) - \hat{v}(\mathbf{s}_t, \mathbf{w}_t), \\ \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha^{\mathbf{w}} \delta_t \nabla \hat{v}(\mathbf{s}_t, \mathbf{w}_t). \end{aligned} \tag{5.22}$$

During planning time the agent performs a regular planning update for $k$ planning steps. We refer to this strategy as *Decide-Time Planning* (DTP).

One may think of this method as a one-step lookahead at the time the action needs to be taken by the agent. This method does not maintain an explicit action-value function, yet it allows the agent to select an action online during decision time as if it were to have one, because it uses the output from the model. In this approach the backup values play the same role as action-values in greedy action selection with an action-value method.

The equation (5.21) is greedy action selection with respect to the model when there is a model for each of the actions. In practice, one may want to ensure that the agent is trying actions that improve the model—often referred to as *exploration* (Sutton and Barto, 2018, Chapter 8). One way to do this is to change (5.21) to $\epsilon$-greedy action selection in which with a small probability $\epsilon$ a random action is selected among all the

actions with equal probability, independently of the value function estimates. There are other methods that ensure exploration that are out of the scope of this thesis.

We show the DTP strategy algorithmically at the end of this section.

### 5.5.3 Strategy 2: Adjunct Action-value Function

Here, we discuss the second strategy for action selection when planning with the state-value function. In this strategy, we maintain an *adjunct action-value function* approximator $q(\mathbf{s}, a, \mathbf{w}^q)$ parameterized by the weight vector $\mathbf{w}^q \in \mathbb{R}^d$. However, this action-value function is not used in AAVI and it is not used to compute the ZTEM TD target $b(\mathbf{s}_t, a, \mathbf{w}_t)$. During decision time, the agent uses the action-value function $q(\mathbf{s}, a, \mathbf{w}^q)$ to act greedily:

$$A_t \doteq \arg\max_{a \in \mathcal{A}} \hat{q}(\mathbf{s}_t, a, \mathbf{w}_t^q),$$

then the agent uses a regular TD update (5.22) and also updates the adjunct action-value function parameters $\mathbf{w}^q$ with the step size $\alpha^q$:

$$\mathbf{w}_{t+1}^q \doteq \mathbf{w}_t^q + \alpha^q \Big( R_{t+1} + \gamma \mathbf{w}^\top \mathbf{s}_{t+1} - \hat{q}(\mathbf{s}_t, a, \mathbf{w}^q) \Big) \nabla \hat{q}(\mathbf{s}_t, a, \mathbf{w}_t^q).$$

During planning time the agent performs a regular planning update for $k$ planning steps and, in addition, it also updates the adjunct action-value function parameters using the ZTEM TD target $b(\mathbf{s}_t, a, \mathbf{w}_t)$:

$$\mathbf{w}_{t+1}^q \doteq \mathbf{w}_t^q + \alpha^q \Big( b(\mathbf{s}_t, a, \mathbf{w}_t) - \hat{q}(\mathbf{s}_t, a, \mathbf{w}^q) \Big) \nabla \hat{q}(\mathbf{s}_t, a, \mathbf{w}_t^q). \tag{5.23}$$

We refer to this strategy as *Adjunct Action-value Function* (Adjunct Q).

Maintaining the adjunct action-value function avoids the problem of the invalid update that we stated in Section 5.4. At the same time, the Adjunct Q method takes less time during decision time compared to the DTP method because the agent no longer needs to calculate the values during decision time. We show the Adjunct Q strategy algorithmically at the end of this section.

### 5.5.4 Strategy 3: Adjunct Policy

Here, we discuss the third strategy for action selection when planning with the state-value function. In this strategy, we maintain an *adjunct policy* approximator $\pi(\mathbf{s}, a, \boldsymbol{\theta})$ parameterized by the weight vector $\boldsymbol{\theta}$. During decision time, the agent uses the policy $\pi(\mathbf{s}, a, \boldsymbol{\theta})$ to act:

$$A_t \sim \pi(\cdot \mid \mathbf{s}_t, \boldsymbol{\theta}_t),$$

then the agent uses a regular TD update (5.22) and also updates the policy parameters $\boldsymbol{\theta}$ with the step size $\alpha^\pi$ as in the regular actor update :

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha^\pi \delta_t \nabla \ln \pi(\mathbf{s}_t, a, \boldsymbol{\theta}_t).$$

During planning time the agent performs a regular planning update for $k$ planning steps and, in addition, it also updates the adjunct policy parameters using the ZTEM TD target $b(\mathbf{s}_t, a, \mathbf{w}_t)$:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha^\pi \Big[ b(\mathbf{s}_t, a, \mathbf{w}_t) - \mathbf{s}^\top \mathbf{w} \Big] \nabla \ln \pi(\mathbf{s}_t, a, \boldsymbol{\theta}_t).$$

We refer to this strategy as *Adjunct Policy* (Adjunct $\pi$).

This third strategy is similar to Adjunct Q, but instead of adjunct action values we have an adjunct explicit approximate policy. Having the approximate policy can be useful in problems when the action space is not too large. Parameterized policies enable the selection of actions with arbitrary probabilities and may be simple functions to approximate (Sutton and Barto, 2018, Chapter 13). In this method actions are sampled from the parameterized policy at decision time and the policy is updated as in policy gradient methods. The difference from policy gradient methods is that during planning time we update the policy parameters based on the output of the model. We show the Adjunct $\pi$ strategy algorithmically at the end of this section.

Here we show all the three methods—DTP, Adjunct Q, and Adjunct $\pi$—algorithmically.

---

**Algorithm:** Decide-Time Planning (DTP)

**Algorithm parameters:** step size $\alpha$

Initialize the weights $\mathbf{w}$ for state-value function, and weights for the model

Compute state $\mathbf{s}$ with the state-update function $u$

**while** *still time to train* **do**

  $A \sim \arg\max_{a \in \mathcal{A}} \left[ r(\mathbf{s}, a) + \gamma \bar{\mathbf{s}}(\mathbf{s}, a)^\top \mathbf{w} \right]$  (e.g., $\epsilon$-greedy) (this is a lookahead)

  Take action $A$, observe $R, O'$

  Compute state $\mathbf{s}'$ with the state-update function $u$

  $\delta \leftarrow R + \gamma \mathbf{w}^\top \mathbf{s}' - \mathbf{w}^\top \mathbf{s}$

  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla (\mathbf{w}^\top \mathbf{s})$

  Update model weights

  **while** *still time to plan* **do**

    Sample state $\mathbf{s}$ from the backup distribution

    $g(\mathbf{s}, \mathbf{w}) \leftarrow \max_{a \in \mathcal{A}} \left[ r(\mathbf{s}, a) + \gamma \mathbf{w}^\top \bar{\mathbf{s}}(\mathbf{s}, a) \right]$

    $\mathbf{w} \leftarrow \mathbf{w} + \alpha (g(\mathbf{s}, \mathbf{w}) - \mathbf{w}^\top \mathbf{s}) \nabla (\mathbf{w}^\top \mathbf{s})$

  **end**

  $\mathbf{s} = \mathbf{s}'$

**end**

---

**Algorithm:** Adjunct Action-value Function (Adjunct Q)

**Algorithm parameters:** step size $\alpha$, $\alpha_q$

Initialize the weights $\mathbf{w}$, $\mathbf{w}^q$ for value functions, and model weights

Compute state $\mathbf{s}$ with the state-update function $u$

**while** *still time to train* **do**

  $A \sim \arg\max_{a \in \mathcal{A}} \hat{q}(\mathbf{s}, a, \mathbf{w}^q)$ (e.g., $\epsilon$-greedy)

  Take action $A$, observe $R, O'$

  Compute state $\mathbf{s}'$ with the state-update function $u$

  $\delta \leftarrow R + \gamma \mathbf{w}^\top \mathbf{s}' - \mathbf{w}^\top \mathbf{s}$

  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla (\mathbf{w}^\top \mathbf{s})$

  $\mathbf{w}^q \leftarrow \mathbf{w}^q + \alpha^q \Big( R + \gamma \mathbf{w}^\top \mathbf{s}' - \hat{q}(\mathbf{s}, a, \mathbf{w}^q) \Big) \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}^q)$

  Update model weights

  **while** *still time to plan* **do**

    Sample state $\mathbf{s}$ from the backup distribution

    Output next state $\bar{\mathbf{s}}$ using the model

    $g(\mathbf{s}, \mathbf{w}) \leftarrow \max_{a \in \mathcal{A}} \big[ r(\mathbf{s}, a) + \gamma \mathbf{w}^\top \bar{\mathbf{s}}(\mathbf{s}, a) \big]$

    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big( g(\mathbf{s}, \mathbf{w}) - \mathbf{w}^\top \mathbf{s} \big) \nabla (\mathbf{w}^\top \mathbf{s})$

    // Update cached action-value function

    $b(\mathbf{s}, a, \mathbf{w}) \leftarrow r(\mathbf{s}, a) + \gamma \bar{\mathbf{s}}(\mathbf{s}, a)^\top \mathbf{w}$

    $\mathbf{w}^q \leftarrow \mathbf{w}^q + \alpha_q \Big( b(\mathbf{s}, a, \mathbf{w}) - \hat{q}(\mathbf{s}, a, \mathbf{w}^q) \Big) \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}^q)$

  **end**

  $\mathbf{s} = \mathbf{s}'$

**end**

**Algorithm:** Adjunct Policy (Adjunct $\pi$)

**Algorithm parameters:** step size $\alpha_{\mathbf{w}}$, $\alpha_{\theta}, \alpha_{\boldsymbol{\eta}}$

Initialize the weights $\mathbf{w}$, $\boldsymbol{\theta}$ for the value function and policy, and model weights

Compute state $\mathbf{s}$ with the state-update function $u$

**while** *still time to train* **do**

  $A \sim \pi(\mathbf{s}, a, \boldsymbol{\theta})$ (e.g., $\epsilon$-greedy)

  Take action $A$, observe $R, O'$

  Compute state $\mathbf{s}'$ with the state-update function $u$

  $\delta \leftarrow R + \gamma \mathbf{w}^\top \mathbf{s}' - \mathbf{w}^\top \mathbf{s}$

  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla(\mathbf{w}^\top \mathbf{s})$

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\pi \delta \nabla\big(\log \pi(\mathbf{s}, a, \boldsymbol{\theta})\big)$

  Update model weights

  **while** *still time to plan* **do**

    Sample state $\mathbf{s}$ from the backup distribution

    $g(\mathbf{s}, \mathbf{w}) \leftarrow \max_{a \in \mathcal{A}} \big[r(\mathbf{s}, a) + \gamma \mathbf{w}^\top \bar{\mathbf{s}}(\mathbf{s}, a)\big]$

    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(g(\mathbf{s}, \mathbf{w}) - \mathbf{w}^\top \mathbf{s}\big)\nabla(\mathbf{w}^\top \mathbf{s})$

    // Update policy parameters

    Sample action $a$ from $\pi_\theta(\mathbf{s}, \cdot)$

    $b(\mathbf{s}, a, \mathbf{w}) \leftarrow r(\mathbf{s}, a) + \gamma \bar{\mathbf{s}}(\mathbf{s}, a)^\top \mathbf{w}$

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\pi \Big[b(\mathbf{s}, a, \mathbf{w}) - \mathbf{s}^\top \mathbf{w}\Big] \nabla \ln\, \pi(\mathbf{s}, a, \boldsymbol{\theta})$

  **end**

  $\mathbf{s} = \mathbf{s}'$

**end**

## 5.6 Non-stationarity Setting & Impact

To demonstrate the action selection strategies, we first introduce an additional setting. In this section we describe a change we introduced to the stochastic corridor to make the environment non-stationary and demonstrate the impact of the change. Understanding the impact of non-stationarity is necessary to further interpret the results in the following sections. More specifically, inspired by the work of van Seijen et al. (2020) that measured how quickly an reinforcement learning method adapts to a local change in the environment, we introduced a goal change to the stochastic corridor. We refer to this setting as the *non-stationary setting*. In this setting the goal location changes every $n$ episodes. We refer to $n$ episodes as *a phase*. At the end of each phase the other terminal state becomes the goal state. For example, state 1 becomes a goal state and the agent would now receive +20 when it goes to the left of the corridor and 0 when the agent goes to the right of the corridor.

### 5.6.1 Experiments

Here, we show the impact of non-stationarity. The implementation was done using PyTorch 1.8.1. We used the discount parameter $\gamma = 1$. All weights (weight vectors) were initialized to zeros. We used a replay buffer for the backup distribution in planning. For all methods, we performed parameter studies during which we ran the algorithms for 1,000 episodes. The results for each of the parameter sets were averaged based on 30 runs, each begun with a different random number seed. The final parameters were chosen based on the best performing parameter set. For each episode, we measured the total reward per episode, $G_0$.

We used tabular-features, meaning states were encoded as *one-hot vectors*. We used

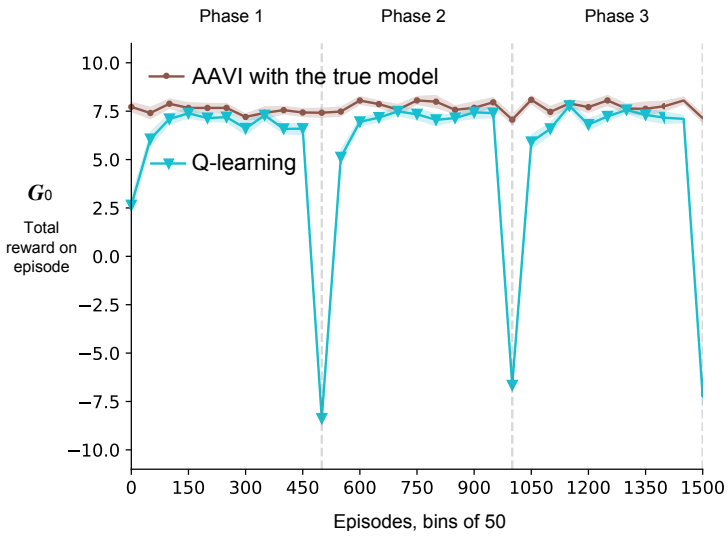


Figure 5.5: Performance of Q-learning and AAVI with the true model on the stochastic corridor. Total reward per episode, averaged over 30 runs; each data point represents the average number of total reward per episode, averaged over all the runs and over temporal stretches of 50-episode bins. Shaded regions are standard error. The phase was set to $n = 500$ episodes. Stochasticity of the environment was $\zeta = 0.3$.

0.1 for the action-values step size in both Q-learning and AAVI with the expectation model. We used 0.1 for the expectation model step-size. We set the number of planning steps $k = 20$ and the stochasticity of the environment was $\zeta = 0.3$. The model was updated as in (5.14).

### 5.6.2 Results

This section describes the performance of Q-learning, AAVI with the true model, and AVI with a learned expectation model on the non-stationary stochastic corridor. Figure 5.5 shows the Q-learning method and AAVI with the true model on the stochastic non-stationary corridor with the phase $n = 500$ episodes. We measured the total re-

ward per episode from the start of the episode, $G_0$, averaged over 30 runs and then averaged over bins of 50 episodes. The value at the phase switch represents the average over the first 50 episodes of the following phase. The total reward per episode when following the optimal policy was about 7.5 because we increased the stochasticity of the environment from $\zeta = 0.1$ to $\zeta = 0.3$.

The first obvious feature of this graph is that the reward dramatically drops at the phase transition from one to another. This was expected because the agent using Q-learning kept going in the wrong direction right after the goal changed. As a result, Q-learning obtained around -7.5 in total reward around the first 50 episodes of phase 2.

Q-learning took a relatively long time to find the goal state consistently every time the goal was moved during a phase. However, the Q-learning method performed a bit better at the second phase transition suggesting that it may have already learned something about the phase transition. Initially, the value for Q-learning was about 2.5 before the phase transition: Q-learning performed better when started with zero knowledge about the environment compared to when it learned to go to the right side and the goal location was changed. AVVI with the true model adjusted to the phase transition almost instantly. There was no drop in performance on the first phase transition and only a small drop on the second phase transition. Such a quick recovery of model-based methods was expected in this scenario.

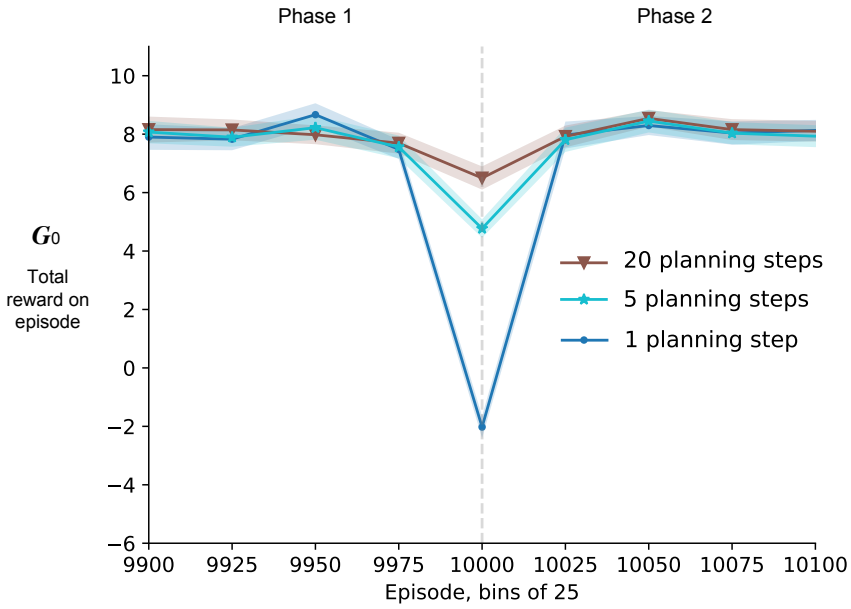


Figure 5.6: AVI with the learned expectation model on the stochastic corridor with tabular features. Higher number of planning steps helped the agent to adjust during the phase transition. A phase was 10,000 episodes. Each data point was the return per episode $G_0$, averaged over 30 runs and over temporal stretches of 25-episode bins. Shaded regions are standard error.

Figure 5.6 shows the AVI method with the learned expectation model on non-stationary stochastic corridor with a phase of 10,000 episodes, with 1, 5, and 20 planning steps. Each data point was the return per episode, $G_0$, averaged over 30 runs and over temporal stretches of 25-episode bins. The graph shows the effect of the number of planning steps: the recovery during the phase transition happened more quickly with the agent using more planning steps. Further, the drop was not as dramatic as we increased the number of planning steps, suggesting that the agent with a higher number of planning steps learned much more quickly during the first 25 episodes after the phase transition. The higher the number of planning steps, the more efficient the agent was at recovering after the phase transition.

## 5.7 Action Selection Strategies Illustration

This section describes the performance of Q-learning (see Section 2.4) and the three action selection strategies, DTP, Adjunct Q, and Adjunct $\pi$, applied to the stochastic non-stationary corridor shown on Figure 5.3, in the function approximation setting.

### 5.7.1 Experiments

The implementation was done using PyTorch 1.8.1. In all experiments we used the following parameters. The discount parameter $\gamma$ was 1. All weights were initialized to zeros. The backup distribution in planning used a replay buffer. For all methods, we performed parameter studies during which we ran algorithms for 1,000 episodes. The results for each of the parameter sets were averaged based on 30 runs, each begun with a different random number seed. The final parameters were chosen based on the best performing parameter set. For each episode, we measured the total reward per episode, $G_0$.

In these experiments on the stochastic corridor, the feature vectors for each state are random binary feature vectors of size $d = 14$, with 5 bits picked at random, without replacement. The phases were 500 episodes. We used $\epsilon$-greedy action selection during decision time, with $\epsilon = 0.1$. We used 0.1 for the action-values step size in both Q-learning and AAVI with the expectation model. The model step size was set to 0.1, 0.001, and 0.01 for DTP, Adjunct Q, and Adjunct $\pi$ respectively. The state-value function step size was set to 0.01 for all three algorithms: DTP, Adjunct Q, and Adjunct $\pi$. We used 0.01 for the adjunct action-values step size in Adjunct Q. We set the number of planning steps to $k = 5$ and the stochasticity of the environment was $\zeta = 0.1$. We used $\epsilon$-greedy action selection with $\epsilon = 0.1$. The adjunct policy step size

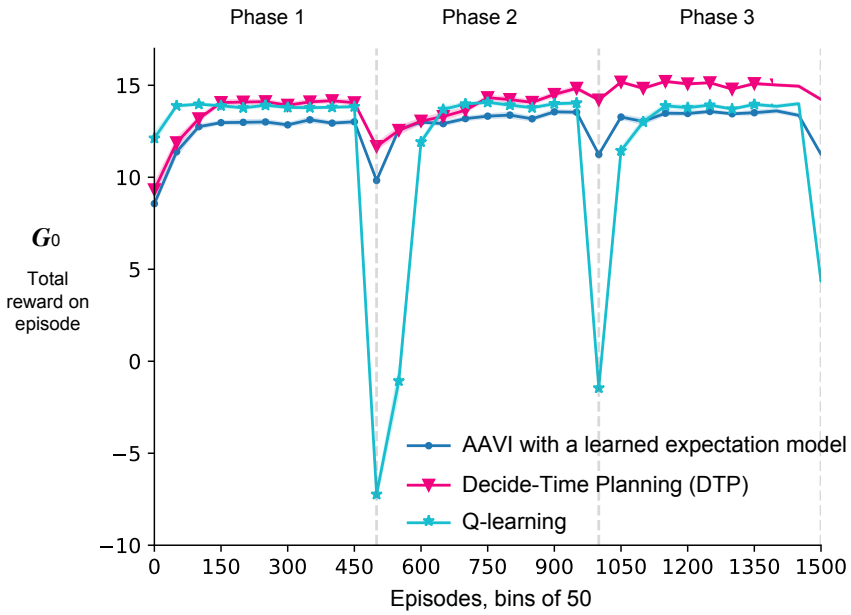


Figure 5.7: Q-learning, AAVI, and DTP on the non-stationary stochastic corridor with a phase of $n = 500$ episodes, with $k = 5$ planning steps. Each data point was the episode return, $G_0$, averaged over all 30 runs and over temporal stretches of 50-episode bins. Shaded regions are standard error. The stochasticity of the environment was $\zeta = 0.1$.

was set to 0.000005 in Adjunct $\pi$. In all these experiments the model was a learned model. The model was updated as in (5.14).

### 5.7.2 Results

Figure 5.7 shows the Q-learning, AAVI with the learned expectation model, and the DTP methods on the non-stationary stochastic corridor. Each data point was the episode return, $G_0$, averaged over 30 runs and over temporal stretches of 50-episode bins. The stochasticity of the environment was $\zeta = 0.1$. We used $\epsilon$-greedy for action

selection with $\epsilon = 0.1$.

Similar to some of the earlier experiments, the first obvious feature of this graph is that the reward dramatically drops at the phase transition. Right after the goal changed, Q-learning obtained around -7.5 in total reward around the first 50 episodes of the phase 2, similar to what we have seen already in Figure 5.5. Notably, at every following phase transition Q-learning performed better and better: around -2.5 at the second phase transition and around 2 at the third phase transition.

DTP performed better overall, and was close to the optimal policy at around 14.5. The drop for DTP was significantly smaller than for Q-learning and the recovery was much faster.

AAVI with the learned expectation model did not reach optimal performance at all, which was expected based on our earlier illustrations of the incompatibility of planning with expectation models and action-value functions in stochastic environments. Interestingly, during the phase transition this method did not experience as large of a drop as Q-learning and also recovered faster, yet to a suboptimal policy.

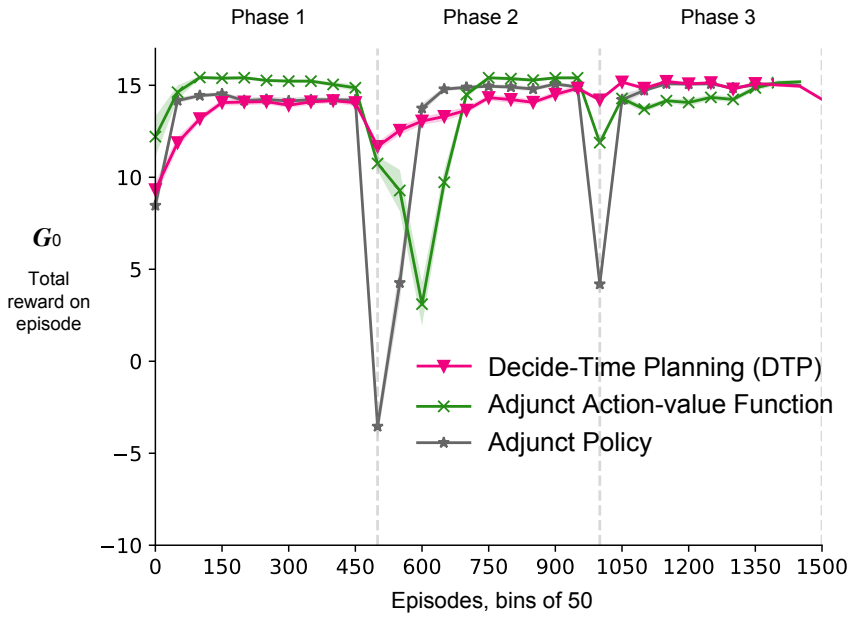


Figure 5.8: DTP, Adjunct Q, and Adjunct $\pi$ methods on the non-stationary stochastic corridor with a phase of $n = 500$ episodes, with $k = 20$ planning steps. Each data point was the episode return, $G_0$, averaged over 30 runs and over temporal stretches of 50-episode bins. Shaded regions are standard error. The stochasticity of the environment was $\zeta = 0.1$.

Figure 5.8 shows the performance of the three action selection strategies on the non-stationary stochastic corridor. Figure 5.8 shows the performance of the DTP, Adjunct Q, and Adjunct $\pi$ methods on the non-stationary stochastic corridor with a phase of $n = 500$ episodes, with $k = 20$ planning steps. Each data point was the episode return, $G_0$, averaged over 30 runs and over temporal stretches of 50-episode bins. The stochasticity of the environment was $\zeta = 0.1$. We used $\epsilon$-greedy for action selection with $\epsilon = 0.1$.

Consistent with the previous results, the performance of all three methods dropped during phase transitions. However, for each of the methods, their recovery after the

second phase transition was shorter than after the first phase transition.

During the first phase, DTP and Adjunct Q obtained similar performance. Adjunct Q performed better than both DTP and Adjunct $\pi$ in this phase. However, this behaviour changed in the second and third phases. In the second phase, Adjunct $\pi$ performed better than DTP, but Adjunct Q still performed the best at the end of this phase. However, in the third phase, Adjunct Q performed slightly worse in the beginning of the phase than the other two methods.

Notably, Adjunct $\pi$ had the largest drop to almost -4 during the phase transition across the three methods. Adjunct Q dropped to around 11 in the first phase transition and to only around 12 during the second one. However, Adjunct Q dropped even lower to around 2 at the beginning of the second phase. DTP had the smallest drops across all three methods: it dropped to only around 12 during the first phase transition and had only a minor drop to around 14 during the second phase transition.

## 5.8 Discussion

In the first set of experiments in Section 5.6.2 we observed that non-stationarity can have a dramatic impact on the performance. In our experiments, the model-based method recovered faster than model-free Q-learning, which is expected. A larger number of planning steps clearly benefits the agent's performance.

Section 5.7.2 illustrates our action selection strategies on this non-stationary domain, now in the setting of function approximation. The experiment settings are such that in all cases, at the beginning of the second phase the agent starts from a value function that induces the agent to move in one direction opposite to the goal. However, at the beginning of the third phase the agent no longer starts from the value function

that moves the agent to the opposite direction. The value function now is no longer the optimal value function from the previous phase because it includes the learning from both of the previous phases that are the opposite of each other. This could explain the slightly shorter recovery period in the third phase transition.

A disadvantage of DTP is that it involves significant computation at decision time, because it has to compute one-step backups for all actions. This could add significant latency to the agent's responses during decision time, especially if the number of actions to be considered is large. This could become an issue in real-life applications such as voice document editing. Adjunct Q and Adjunct $\pi$ mitigate this issue to an extent because they use an explicit adjunct action-value function and an explicit policy respectively. The adjunct action-value function and the adjunct policy help to shorten the computation during the decision time because the agent simply can use the existing action-value function or policy to act. Another strength of Adjunct Q compared to DTP is its flexibility with regards to when the action values are updated—for which states and actions (5.23) is performed. Adjunct $\pi$ has a similar flexibility in terms of updating the policy. On the other side, Adjunct Q and Adjunct $\pi$ have more parameters compared to DTP. A larger number of parameters can be a disadvantage because it takes longer to find the best performing parameters and also comes at a higher computational cost.

## 5.9 Conclusion and Future Work

Many problems considered in AI exhibit non-stationarity and dynamically changing goals paired with some level of stochasticity in the domain. Expectation models are appealing and have benefits compared to sample models and distribution models. Expectation models are expected to have less variance and hence result in quicker learning. In addition, their space and computational requirements are smaller, and they may require less parameter tuning.

One may think that when planning with expectation models AAVI is a suitable approach. We showed in this chapter that planning with expectation models cannot be done with action-value functions and must update state values in stochastic environments. This raised an important question of action selection when planning and we considered three action selection strategies. We demonstrated these action selection strategies empirically. The studies reported here represent a general setting of model-based reinforcement learning with expectation models that are independent of many choices such as that of state representation. The results show how planning expectation models with function approximation can proceed in stochastic environments when using state-value functions.

Interesting areas for future work lie in a few directions: 1) how different kinds of backup distributions affect planning; 2) how the degree of function approximation affects planning; and 3) comparison of one-step backups in expectation models and iterated backups in sample models, and 4) further extension of the theory in this chapter to the case of general non-linear value functions.

# Chapter 6

# Application for VDE and Beyond

The role of this chapter is to connect the main contribution of the thesis with the first contribution of the thesis. In this final chapter, we move our attention from the general theory back to the practical application of this theory and our ideas on our VDE domain. We start with an experiment that takes the theory from Chapter 5 and applies the action selection strategies to the VDE domain. Finally, we discuss two questions that are important to think about when implementing the voice document-editing domain in practice: 1) How ambitious can the agent's learning be? and 2) Should intelligent assistants learn online or offline?

## 6.1 Action Selection Strategies Applied to VDE

In this section, we describe the experiments and results of applying the action selection strategies to the VDE task. Recall that the VDE task represents an episodic control problem with a stochastic environment in which the agent learns online. In this task

the agent receives a corrupted sentence and has to correct it by deleting a number of words. We want the agent to delete the correct number of corrupted words in the fewest actions possible, which would reduce the interaction with the user during the deletion process, and as a result reduce users' frustration.

### 6.1.1 Experiments

In these experiments, we compared the performance of the DTP, Adjunct Q, and Adjunct $\pi$ methods. We measured the total reward per interaction, $G_0$, averaged over 30 runs, each begun with a different random number seed.

We proceed with our defined function approximation setting, as in Chapter 4. The state-update function architecture for the voice document-editing domain remained the same as described in Section 4.4.2; the state-update function $u$ was represented by a two-layered bi-directional RNN with hidden sizes of 400 and 100, respectively. The RNN-input was computed using the same approach as in Chapter 4 (see Figure 4.2). We also used the same word embeddings described in Section 4.4.1.

The implementation was done using PyTorch 1.8.1. Similar to the implementation in Section 4.4.5, a starting state for the state update function was initialized to ones and a starting action to perform the first state-update iteration was initialized to zeros. We used $\epsilon$-greedy action selection during decision time, with $\epsilon = 0.1$.

For all methods, we performed parameter studies during which we ran algorithms for 1,000 episodes. The results for each of the parameter sets were averaged based on 30 runs, each begun with a different random number seed. The final parameters were chosen based on the best performing parameter set. The state-value function step size was set to 0.001, 0.001, and 0.005 for DTP, Adjunct Q, and Adjunct $\pi$ respectively.

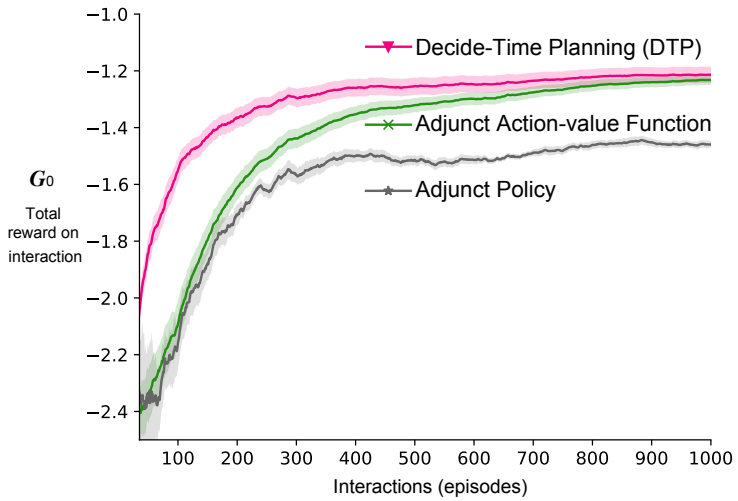


Figure 6.1: Three action selection strategies the voice document-editing deletion task (see Section 3.6.1). We measured total reward per interaction, $G_0$, averaged over 30 runs, each begun with a different random number seed. Shaded regions are standard error.

The adjunct action-value function step size was set to 0.001 in Adjunct Q. The model step size was set to 0.001 for all three methods: DTP, Adjunct Q, and Adjunct $\pi$. The adjunct policy step size was set to 0.001 in Adjunct $\pi$. The number of planning steps was set to $k = 10$.

### 6.1.2 Results

Figure 6.1.2 shows the performance of the action selection strategies on the VDE task—total reward per interaction, $G_0$, averaged over 30 runs, each begun with a different random number seed. The first feature of this graph is that all three methods reached a performance of over -1.5 on average. This means that the agent trained with these methods would correct a sentence with 1-1.5 mistakes on average. DTP

reached a performance of around -1.2 more quickly than the other two methods. It also showed the best overall performance of all three. Adjunct Q caught up with DTP after approximately 700 interactions. Adjunct $\pi$, however, started similar to Adjunct Q, but then stayed at around -1.5. Overall, the final performance (at 1,000 interactions) of DTP and Adjunct Q is almost the same, and the final performance of all three methods was quite close to each other. The interpretation of the results is that DTP and Adjunct Q took on average around 1 mistake to correct the sentence at the end of training, while Adjunct $\pi$ took 1.5 mistake on average.

### 6.1.3 Conclusions

Generally, we can say that all three action selection strategies performed well on the VDE task—averaging close to 1 mistake in making a sentence correction is close to the optimal performance. Adjunct $\pi$ potentially is more sensitive because it has an explicit policy that could be getting stuck in local optima. Also, Adjunct $\pi$ has more parameters to finetune and it is possible that there could be another set of parameters that leads to better performance. DTP had the best performance amongst the three, however, this could change with other parameters.

In these experiments, we empirically demonstrated the application of the action selection strategies to the VDE task. We showed that these methods could perform well and can be implemented for this real-life application.

## 6.2 Some Remaining Questions

There are many open questions when it comes to the implementation of systems such as VDE. This thesis cannot cover all of them. Here, we discuss two questions specific to our voice document-editing domain: 1) How ambitious can the agent's learning be? and 2) Should intelligent assistants learn online or offline? These questions are important, because the answers to them have a ripple effect to other implementation choices such as representation possibilities and a selection of models.

The first question is how ambitious can the agent's learning be? Recall that there are many complex elements in conversational AI, such as natural language understanding, dialogue management, natural language generation, and response generation. For example, the agent's learning can focus on only one (or on a few) of the elements, while the remaining elements can be treated as a black box; for example, the user's speech can be a black box and can be transcribed by existing tools for speech-to-text conversion (e.g., Bijl and Hyde-Thomson, 2001; Rao, 2011). When not treated as a black box, each of the elements has its own challenges; for example, diversity and coherence in natural language generation (e.g., Yarats and Lewis, 2017; Jang et al., 2019; Shi et al., 2018; Gu et al., 2019; Wang et al., 2019). The agent's ambitiousness in learning can vary in its complexity with the choices for and within each element of conversational AI.

The agent's learning within one element in conversational AI can be thought of as having three levels of complexity. As an example, consider natural language generation. The first level of complexity is when the agent's responses are entirely pre-defined by a system designer; this means that no language generation occurs. The agent's learning then focuses only on the selection of actions to satisfy the user, and the agent's only

intelligence is in the learned policy that it follows to select a response. The model could then learn the dynamics of the interactions and still help the agent with this action selection, e.g., when using planning. Despite the absence of language generation, this agent can still be useful to its users. The second level of complexity is when the response generation is treated as a black box and can be supplemented by one of the existing approaches (e.g., Serban et al., 2017b) instead of using pre-defined responses as in the first level. The third level of complexity is when the response generation is fully learned and this learning becomes a part of the model-based reinforcement learning agent. The response generation is a demonstration of the agent's ambitiousness in learning within one element of conversational AI.

Our answer to the first question is that an ambitious agent is the most desirable; the agent can be fully responsible for learning all the elements of conversational AI, from generating responses to selecting them. A model will be more complex for this sophisticated agent because it will have to learn all about the interaction dynamics between the user and the assistant. These kinds of implementations are often referred to as end-to-end training. Variations of such implementations can include multiple agents with multiple models, where each of the agent-model pairs can be responsible for a particular conversational AI element. Further, there are higher-level models of the world based on temporally extended abstract behavior, which were introduced as *options* by Sutton et al. (1999). Higher-level models provide a way of obtaining higher-level planning and reasoning (Sutton, 2020).

The second question is how to train voice document-editing assistants: should they learn online or offline? Recall that online learning is learning directly from experience, such as from real human-computer interactions. Offline learning is a traditional approach that is often used in supervised learning that relies on pre-constructed datasets.

These datasets can be constructed in a number of ways, including real user-computer interactions, and can be used then to build simulators.

One way an offline dataset can be created is by using the Mechanical Turk service, a crowdsourcing web service that coordinates the supply and demand of tasks (see Paolacci et al., 2010). Using Mechanical Turk, dictation and editing of the resulting documents can be recorded as a dataset, which is then used to create an environment for voice editing simulations. As Dhingra et al. (2017) pointed out, it is common in the dialogue community to use simulated users for this purpose (e.g., Schatzmann et al., 2007; Cuayáhuitl et al., 2005; Asri et al., 2016).

Another way to obtain offline datasets requires more creative approaches. Consider the work of Feng et al. (2019) in which organizational business documents were used as inputs to generate a conversational offline dataset. In this conversational dataset, the conversation is based on the organization's workflow. We encourage the reader to think about similar creative ways of obtaining document-editing datasets. For example, with some good engineering, the mechanical manipulations of text blocks in manuscripts can be collected and converted into conversational data with the addition of voice commands.

Our preferred method of training would be online learning from real interaction data. Online learning allows the intelligent assistant to adapt to each individual and their particular circumstances during the training process, and it creates an opportunity for the assistant and the user to build the communication resources developed together during their ongoing interaction (see Pilarski et al., 2017). Building communication resources can effectively improve collaboration and interaction between the user and the voice-editing assistant. For example, a person with an accent might be consistently misinterpreted by the voice recognition feature, and the voice-editing assistant could

end up having to delete the same words over and over to fix the sentence. If the assistant is trained offline on many other people's data then it might do worse at recognizing and fixing such an individual situation, thus its quallity would be worse overall. As Mendez et al. (2019) point out, a widespread adoption by users of intelligent assistants is limited to the assistants' quality, which often requires the investment of vast amounts of data. Thus, even if we were to have access to systems that allow us to experiment with real online data (e.g., Google, 2020), then the assistant that learns fully online might be of lower quality in the beginning of training than one that was pre-trained with offline datasets. In the past, when a secretary was hired, they were expected to have structural language knowledge. Similarly, it is reasonable to have such expectations of voice editing assistants and pre-train them with some preliminary knowledge learned from offline datasets before these assistants can be offered to users and continue to learn online.

# Chapter 7

# Impact and Summary

In this thesis, we started with a discussion of a problem that arises when developing conversational AI assistants—the need for a suitable but tightly scoped domain. In Chapter 3 we proposed the voice document-editing domain to address the problem. We then discussed reinforcement learning, and in particular, model-based reinforcement learning methods as suitable methods for developing intelligent assistants and for realizing VDE. In Chapter 4 we demonstrated the application of model-based reinforcement learning methods in practice—we proposed the model-based, soft-planner policy optimization method for solving voice document editing. Our work in Chapter 4 motivated us to explore deeper planning with expectation models in function approximation settings and seek methods that are applicable not only for solving VDE, but for building intelligent assistants in general. It is natural to think that planning can be done with action-value functions. However, in Chapter 5 we proved the incompatibility of planning with expectation models and action-value functions. This result raised an important question of how to select actions when planning. We considered three action selection strategies that are applicable to general settings. We empirically

demonstrated these strategies and discussed their advantages and disadvantages. Finally, in Chapter 6 we demonstrated these action selection strategies empirically on the VDE domain.

This thesis is a long journey from a high-level general problem to this specific specific question of action selection when planning with expectation models in function approximation settings and stochastic environments; a question that came from examining more closely planning with action-value functions and proving their incompatibility in these settings. The incompatibility of expectation models and action-value functions and the action selection strategies that we showed in Chapter 5 are applicable to general settings and we hope that this work will serve as a basis for the implementation of intelligent assistants and for further investigation of planning methods.

## 7.1 Future Work

Broadly, in this thesis we looked at model-based reinforcement learning methods that use planning and expectation models. We applied these methods in the context of intelligent assistants, in particular, voice document-editing assistants. While there is progress in this direction, there are important challenges that remain. In this section, we outline some remaining open areas of this research direction.

1. **Non-linear value functions.** Our setting includes function approximation, meaning that the agent states are described by feature vectors, and the value function and transition model use function approximators, such as neural networks. There is the usual spectrum of possible approximators, from state aggregation and linear, then semi-linear, and finally general non-linear. In Chapter 5,

we focused on linear value functions for function approximators. There are currently no promising approaches that extend planning with expectation models to non-linear value functions or other more complex approximators.

2. **Temporal abstractions.** Temporal abstractions can speed up agent's learning even further increasing the effectiveness of the training which is most valuable in human-machine interactions (e.g., Fikes, Hart, and Nilsson, 1972; Minton, 1988; Iba, 1989). This work can be extended to temporally abstract courses of action over a period of time—known as *options* (Sutton, Precup, and Singh, 1998, 1999; Precup, 2000). In such, this work can also be extended to option models.

3. **Continuing setting.** The lives of many living creatures, including humans, can be formulated as a continuing problem. Usually, there is a finite amount of time during which a living being interacts with the world, and there is no reset after that. Naik et al. (2021) suggest that we can think about such settings as one long episode. These are settings that do not have a set break for the end of the interaction the—*continuing setting*. In this thesis, we focused on episodic problems. The methods presented could be investigated for the continuing setting and continuing control. Such a setting would change the reward formulation. Recall that the discounted formulation would not be suitable in the case of continuing control with function approximation (Sutton and Barto, 2018, Chapter 10; Naik et al., 2019) and the setting would have to be extended to the average reward formulation. This is an important direction for real-world applications.

4. **Backup distribution & Search control.** Recall that planning can often be described as proceeding in a sequence of state-value backups, each of which updates the value estimate of a single state—the backup state (see Section 5.2.1).

Also, recall that a process for selecting the backup state is referred to as search control. There are many choices available and many different ways the backup state can be selected (e.g., see search-control in Dyna by Pan et al., 2019). However, more sophisticated search control methods could be analysed, e.g., search control by Pearl (1984) or prioritized sweeping (Moore and Atkeson, 1993; Peng and Williams, 1993).

5. **Multi-step backups.** Another direction for investigation is whether multi-step backups are necessary and beneficial. With our focus on expectation models, we do not expect to perform multi-step backups. This is because the expected next state may not correspond to any real observation; Jafferfee (2020) showed that planning from such states can be detrimental. In contrast, sample models are amenable to multi-step backups. However, this can also be detrimental to planning because of compounding errors (Talvitie, 2014; 2017). Multi-step backups can compound errors just as one-step backups can if the backup distribution is poor. We have shown that planning works perfectly well with one-step backups, but have not investigated if greater sample-efficiency can be gained from iterated multi-step backups.

6. **Sample models.** Sample models can be used effectively in stochastic environments (Deisenroth and Rasmussen, 2011; Chua et al., 2018). Extending this work using sample model approaches instead of expectation models is another research direction. The sample model is often much easier to obtain computationally rather than a distribution model. A potential area of future research is how sample models affect the stochasticity of the planning update.

7. **Policy-gradient methods.** Policy-gradient methods have many advantages due

to their ability to learn explicit policy parameters. For example, these methods could learn special levels of exploration or approach deterministic policies asymptotically (Sutton and Barto, 2018, Chapter 13). Konda and Tsitsiklis (2000) claimed that policy-gradient methods in continuous control problems are often more stable than value-based methods, particularly in the function approximation setting. Overall, policy-gradient methods are foundational for many real-life applications with state-of-the-art performance across many domains (Schulman et al., 2015, 2017); however, in the function approximation and model-based reinforcement learning setting these methods are less well understood in some respects, and thus, can be a subject of further research.

All of the above are great challenges, each of which would take significant time and effort to investigate, taking us closer to achieving the goals of conversational AI and the bigger challenge of creating intelligent assistants that will improve the lives of many generations to come.

This thesis is the intersection between deep learning, natural language processing, and reinforcement learning, with the focus on model-based reinforcement learning methods. This thesis presents a view on what domains and methods are required to move conversational AI research forward, contributing to the development of intelligent assistants. It is my hope that this work will inspire and uplift the future of the field, advancing the benefits for all of humanity and driving the arc of human progress.

# References


Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of the 21st Conference on Neural Information Processing Systems (NIPS 2007)*, pp. 1–8.

Ades, S. and Swinehart, D. C. (1986). Voice annotation and editing in a workstation environment. Technical Report. CSL-86-3, XEROX Corporation, Palo Alto Research Center.

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences, 10*(1-2):25–61.

Albus, J. S. (1981). *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH.

Allen, J. F. (1979). A plan-based approach to speech act recognition. Technical Report 131/79, University of Toronto, Department of Computer Science.

Anderson, J. R., Boyle, C. F., and Reiser, B. J. (1985). Intelligent tutoring systems. *Science, 228*(4698):456–462. American Association for the Advancement of Science.

Asri, L.E., He, J., and Suleman, K. (2016). A Sequence-to-Sequence Model for User Simulation in Spoken Dialogue Systems. *INTERSPEECH*.

Atkeson, C. G. and Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of International Conference on Robotics and Automation*. Vol. 4, pp. 3557–3564.

Bagnell, J. A. and Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164) (IEEE)*,

vol. 2, pp. 1615–1620.

Baym, N., Shifman, L., Persaud, C., and Wagman, K. (2019). Intelligent Failures: Clippy Memes And The Limits Of Digital Assistants. *AoIR Selected Papers of Internet Research*.

Bapna, A., Tür, G., Hakkani-Tür, D. Z., and Heck, L. (2017). Towards Zero-Shot Frame Semantic Parsing for Domain Scaling. *INTERSPEECH*.

Barto, A. G., Sutton, R. S., Anderson, C. W. (1983). Neuronlike elements that can solve diffcult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 13(5)*:835–846. Reprinted in J. A. Anderson and E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research*, pp. 535–549. MIT Press, Cambridge, MA, 1988.

Bassiri, M. A. (2011). Interactional Feedback and the Impact of Attitude and Motivation on Noticing L2 Form. *English Language and Literature Studies 1*, 61.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, pp. 908–918.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control, Volume II: Approximate Dynamic Programming*, (4th ed.). Athena Scientific, Belmont, MA.

Bertsekas, D. P., Tsitsiklis, J. N. (1996) *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Biermann, A. W. and Long, P. M. (1996). The composition of messages in speech-graphics interactive systems. In *Proceedings of the 1996 International Symposium on Spoken Dialogue*, pp. 97–100.

Bijl, D. and Hyde-Thomson, H. (2001). *Speech to text conversion.* US Patent 6,173,259. Google Patents.

Bordes, A., Boureau, Y.-L., and Weston, J. (2017). Learning end-to-end goal-oriented dialog. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017).*

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.

Bruce, B. (1975). Belief systems and language understanding. *Trends in Linguistics. Studies and Monographs 19*:113–160.

Buesing, L., Weber, T., Racaniere, S., Eslami, S. M. A., Rezende, D., Reichert, D. P., et al. (2018). Learning and querying fast generative models for reinforcement learning. ArXiv:1802.03006.

Budzianowski, P., Ultes, S., Su, P.-H., Mrkšic, N., Wen, T.-H., Casanueva, I., et al. (2017). Sub-domain modelling for dialogue management with hierarchical reinforcement learning. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pp. 86–92.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. (2018). Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, pp. 8224–8234.

Carberry, S. (1989). Plan recognition and its use in understanding dialog. In *User Models in Dialog Systems*, pp. 133–162. Springer.

Carbonell, J. R. (1971). Mixed-Initiative Man-Computer Instructional Dialogues. Technical Report, Cambridge, Bolt Beranek and Newman.

Card, S., Moran, T., and Newell, A. (1980). Computer text-editing: An information-processing analysis of a routine cognitive skill. *Cognitive Psychology, 12*:32–74.

Cassell, J. (2000). More than just another pretty face: Embodied conversational interface agents. *Communications of the ACM 43*(4):70–78.

Chapman, D. and Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Conference on Artificial Intelligence*, pp. 726–731. Morgan Kaufmann, San Mateo, CA.

Chen, L., Chang, C., Chen, Z., Tan, B., Gašić, M., and Yu, K. (2018). Policy adaptation for deep reinforcement learning-based dialogue management. In *IEEE International Conference on Acoustics, Speech and Signal Processing* (IEEE), pp. 6074–6078.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP-2014)*, pp. 1724–1734.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. (2018). In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, pp. 4754–4765, Montréal, Canada. Curran Associates, Inc.

Coates, A., Abbeel, P., and Ng, A. Y. (2017). Autonomous Helicopter Flight Using

Reinforcement Learning. *Encyclopedia of Machine Learning and Data Mining.*. Boston, MA: Springer US.

Cohen, P. R. (1978). *On Knowing What to Say: Planning Speech Acts*. Ph.D. thesis, University of Toronto, Department of Computer Science.

Croft, W. B., Metzler, D., and Strohman, T. (2010). *Search engines: Information retrieval in practice*. (Vol. 520, pp. 131–141). Reading: Addison-Wesley.

Cuayáhuitl, H., Renals, S., Lemon, O., and Shimodaira, H. (2005). Human-computer dialogue simulation using hidden Markov models. In *Workshop on Automatic Speech Recognition and Understanding, IEEE*, pp. 290–295.

Cuayáhuitl, H., Yu, S., Williamson, A., and Carse, J. (2016). Deep reinforcement learning for multi-domain dialogue systems. In *Workshop on Deep Reinforcement Learning, NIPS*.

Cuayáhuitl, H., Yu, S., Williamson, A., and Carse, J. (2017). Scaling up deep reinforcement learning for multi-domain dialogue systems. In *Proceedings of the International Joint Conference on Neural Networks (IEEE)*, pp. 3339–3346.

Dario, P., Guglielmelli, E., Genovese, V., and Toro, M. (1996). Robot assistants: Applications and evolution. *Robotics Auton. Syst., 18*:225–234.

Deisenroth, M., Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th international conference on Machine learning (ICML-11)*, pp. 465–472. Omnipress.

Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2013). Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence 37*:408–423.

Devlin J., Chang M. W., Lee K., Toutanova K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, pp. 4171–4186.

Dhingra, B., Li, L., Li, X., Gao, J., Chen, Y.-N., Ahmed, F., et al. (2017). Towards end-to-end reinforcement learning of dialogue agents for information access. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 484–495.

Digital Inspiration. Dictation.io. http://dictation.io [Accessed July 27, 2020]

Doll, B. B., Simon, D. A., Daw, N. D. (2012). The ubiquity of model-based reinforcement learning. *Current Opinion in Neurobiology, 22*(6):1–7.

Douglas, H. R. (1999). *Method and apparatus for editing documents through voice recognition*. US Patent 5,875,429, Google Patents.

Duffy, J. (2018). *The Best Dictation Software for 2019*. https://zapier.com/blog/best-text-dictation-software [Accessed July 27, 2020]

Engelbart, D. C. (1962). Augmenting human intellect: A conceptual framework. Technical Report. AFOSR-3223. Stanford Research Institute.

Eric, M. (2020). NeurIPS 2019 ConvAI Workshop Recap. *IEEE Signal Processing Society*. https://signalprocessingsociety.org/ [Accessed July 27, 2020]

Fain, D. C., and Pedersen, J. O. (2006). Sponsored search: A brief history. *Bulletin of the American Society for Information Science and technology, 32*(2), 12–13. Wiley Online Library.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S.

(2018). Model-based value estimation for efficient model-free reinforcement learning. ArXiv:1803.00101.

Feng, S., Fadni, K., Liao, Q. V., and Lastras, L. A. (2019). Doc2Dial: A Framework for Dialogue Composition Grounded in Business Documents. In *Workshop on Document Intelligence, NeurIPS*.

Ferreira, E. and Lefèvre, F. (2015). Reinforcement-learning based dialogue system for human–robot interactions with socially-inspired rewards. *Computer Speech Language 34*, 256–274.

Fikes, R., Hart, P., and Nilsson, N. (1972). Learning and executing generalized robot plans. *Artificial Intelligence, 3*:251–288.

Finin, T. W.; Joshi, A. K.; and Webber, B. L. (1986). Natural language interactions with artificial experts. *Proceedings of the IEEE 74*(7):921–938.

Finnsson, H. and Björnsson, Y. (2008). Simulation-Based Approach to General Game Playing. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, vol. 8, pp. 259–264.

Foerster, J., Assael, I. A., De Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Proceedings of the 30st Conference on Neural Information Processing Systems (NIPS 2016)*, pp. 2137–2145.

Forrester, J. W. (1971). Counterintuitive behavior of social systems. *Theory and Decision 2*(2):109–140.

Fox, M. S., and Smith, S. F. (1984). ISIS—a knowledge-based system for factory scheduling. *Expert systems, 1*(1):25–49. Wiley Online Library.

Freeman, D., Ha, D., and Metz, L. (2019). Learning to Predict Without Looking Ahead: World Models Without Forward Prediction. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NIPS 2019)*, pp. 5380–5391.

Fukushima, K., and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition, 15*(6):455–469.

Jiang, J., Hassan Awadallah, A., Jones, R., Ozertem, U., Zitouni, I., Gurunath Kulkarni, R., et al. (2015). Automatic Online Evaluation of Intelligent Assistants. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 506–516.

Jiang, Z., Mao, X.-L., Huang, Z., Ma, J., and Li, S. (2019). Towards End-to-End Learning for Efficient Dialogue Agent by Modeling Looking-ahead Ability. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*.

Gašić, M., Jurčíček, F., Thomson, B., Yu, K., and Young, S. (2011). On-line policy optimisation of spoken dialogue systems via live interaction with human subjects. In *IEEE Workshop on Automatic Speech Recognition & Understanding, IEEE*, pp. 312–317.

Gašić, M., Breslin, C., Henderson, M., Kim, D., Szummer, M., Thomson, B., et al. (2013a). On-line policy optimisation of Bayesian spoken dialogue systems via human interaction. In *IEEE International Conference on Acoustics, Speech and Signal Processing (IEEE-2013)*, pp. 8367–8371.

Gašić, M., Breslin, C., Henderson, M., Kim, D., Szummer, M., Thomson, B., et al. (2013b). POMDP-based dialogue manager adaptation to extended domains. In *Proceedings of the SIGDIAL 2013 Conference*, pp. 214–222. Metz, France: Association for Computational Linguistics.

Gašić, M., Mrkšic, N., Rojas-Barahona, L. M., Su, P.-H., Ultes, S., Vandyke, D., et al. (2017). Dialogue manager domain adaptation using Gaussian process reinforcement learning. *Computer Speech & Language 45*, 552–569.

Gao, J., Galley, M., Li, L., et al. (2019). Neural Approaches to Conversational AI. In *Foundations and Trends in Information Retrieval 13*(2-3):127–298.Now Publishers, Inc.

Gass, S. M. and Varonis, E. M. (1994). Input, interaction, and second language production. *Studies in Second Language Acquisition 16*, 283–302.

Google (2020). Type with your voice: Edit your document. https://support.google .com/docs/answer/4492226 [Accessed February 22, 2020]

Goyal, P., Niekum, S., and Mooney, R. J. (2019). Using natural language for reward shaping in reinforcement learning. In Kraus, S., (Ed.), *Proceedings of the Twenty-Eighth Interna- tional Joint Conference on Artificial Intelligence*, pp. 2385–2391. Macao, China.

Greyson, A. M., Hokit, J. D., Kaptanoglu, M., Wagner, A. M., and Capps, S. P. (1997). *Method and apparatus for the manipulation of text on a computer display screen.* US Patent 5,666,552. Google Patents.

Griffith, S., Subramanian, K., Scholz, J., Isbell, C. L., and Thomaz, A. L. (2013). Policy shaping: Integrating human feedback with reinforcement learning. In *Proceedings of the 27th Conference on Neural Information Processing Systems (NIPS 2013)*, pp. 2625–2633.

Grignetti, M. C., Hausmann, C., and Gould, L. (1975). An "intelligent" on-line assistant and tutor: NLS-scholar. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, pp. 775–781. ACM.

Grosz, B. (1983). Team: A transportable natural language interface system. In *Proceedings of the 1st Conference on Applied Natural Language Processing*, pp. 38–45. Santa Monica, California: ACM.

Gruber, T. R. and Clark, G. C. (2017). *Dictation that allows editing.* US Patent App. 15/268,215. Google Patents.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep Q-learning with model-based acceleration. In *Proceedings of the 33rd international conference on Machine learning (ICML-16)*, vol. 48, JMLR.org.

Gupta, A., Zhang, P., Lalwani, G., and Diab, M.T. (2019). CASA-NLU: Context-Aware Self-Attentive Natural Language Understanding for Task-Oriented Chatbots. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).*

Ha, D. and Schmidhuber, J. (2018). World models. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, pp. 2451–2463.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., at al. (2018a). Soft actor-critic algorithms and applications. ArXiv:1812.05905.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th international conference on Machine learning (ICML-18)*, pp. 1861–1870. PMLR.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., et al. (2018). Learning latent dynamics for planning from pixels. In *Proceedings of the 35th international conference on Machine learning (ICML-18).*

Handoyo, E., Arfan, M., Soetrisno, Y. A. A., Somantri, M., Sofwan, A., and Sinuraya, E. W. (2018). Ticketing chatbot service using serverless NLP technology. In *Proceedings of the 5th International Conference on Information Technology, Computer, and Electrical Engineering*, pp. 325–330. IEEE.

Hawkins, D. (1968). The nature of purpose. In *Purposive systems: Proceedings of the first annual symposium of the American Society for Cybernetics*, pp. 163–173.

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M. (2015). Deep reinforcement learning with a natural language action space. ArXiv:1511.04636.

Hernandaz, E. and Arkun, Y. (1990). Neural Network Modeling and an Extended DMC Algorithm to Control Nonlinear Systems. In *American Control Conference*, pp. 2454–2459. IEEE.

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. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 3215–3222.

Hester, T. and Stone, P. (2011). Learning and using models. in M. Wiering, and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 111–141. Springer-Verlag, Berlin Heidelberg.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8):1735–1780.

Higashinaka, R., *et al.* (2014). Towards an open-domain conversational system fully based on natural language processing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 928–939.

Holland, G. Z., Talvitie, E. J., and Bowling, M. (2018). The Effect of Planning Shape on Dyna-style Planning in High-dimensional State Spaces. ArXiv:1806.01825.

Hoy, M. B. (2018). Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Medical reference services quarterly 37*(1):81–88. Taylor and Francis.

Huang, M., Zhu, X., and Gao, J. (2020). Challenges in building intelligent open-domain dialog systems. *ACM Transactions on Information Systems (TOIS) 38*(3):1–32.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning, 3*:285–317.

Jafferjee, T. (2020). *Chasing Hallucinated Value: A Pitfall of Dyna Style Algorithms with Imperfect Environment Models*. Master's thesis, University of Alberta.

Jang, Y., Lee, J., and Kim, K.-E. (2019). Bayes-Adaptive Monte-Carlo Planning and Learning for Goal-Oriented Dialogues. In *3rd Workshop on Conversational AI, NIPS 2019*.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NIPS 2019)*.

Kaiser, G. E., Feiler, P. H., and Popovich, S. S. (1988). Intelligent assistance for software development and maintenance. *IEEE software, 5*(3), 40–49.

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. (2020). Model-based reinforcement learning for Atari. In *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*.

Kamm, C. (1995). User interfaces for voice applications. In *Proceedings of the National Academy of Sciences 92*(22), 10031–10037.

Kamthe, S. and Deisenroth, M. (2018). Data-efficient reinforcement learning with probabilistic model predictive control. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, eds. A. Storkey and F. Perez-Cruz (Lanzarote, Spain: PMLR), vol. 84 of *Proceedings of Machine Learning Research*, pp. 1701–1710.

Kaplan, F., Oudeyer, P. Y., Kubinyi, E., and Miklósi, A. (2002). Robotic clicker training. *Robotics and Autonomous Systems, 38*(3-4):197–206.

Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. *Proceedings of the 29th Conference on Neural Information Processing Systems (NIPS 2015)*, pp. 3294–3302.

Knox, W. B. and Stone, P. (2012). Reinforcement learning from simultaneous human and MDP reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1 (AAMAS-2012)*, pp. 475–482.

Kocijan, J., Murray-Smith, R., Rasmussen, C. E., and Girard, A. (2004). Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference (IEEE)*, vol. 3, pp. 2214–2219.

Konda V. and Tsitsiklis J. (2000). Actor-critic algorithms. In *Proceedings of the 13th Conference on Neural Information Processing Systems (NIPS 2000)*, pp. 1008–1014.

Kozierok, R. and Maes, P. (1993). A learning interface agent for scheduling meetings. In *Proceedings of the 1st international conference on Intelligent user interfaces*, pp. 81–88. ACM.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. In *Proceedings of the 6th International Conference on Learning Representations (ICLR 2018)*.

Kuvayev, L. and Sutton, R. S. (1996). Model-based reinforcement learning with an approximate, learned model. In *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, pp. 101–105. Yale University, New Haven, CT.

Lau, T., Wolfman, S., Domingos, P.M., and Weld, D.S. (2001). Learning Repetitive Text-Editing Procedures with SMARTedit. *Your Wish is My Command.*

Lazaridou, A., Pham, N. T., and Baroni, M. (2016). Towards multi-agent communication-based language learning. ArXiv:1605.07133.

Leibfried, F., Kushman, N., and Hofmann, K. (2017). A Deep Learning Approach for Joint Video Frame and Reward Prediction in Atari Games. In *Workshop on Principled Approaches to Deep Learning, ICML*, Sydney, Australia.

Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS 2014)*, pp. 1071–1079.

Levin, E., Pieraccini, R., and Eckert, W. (1997). Learning dialogue strategies within the Markov decision process framework. In *IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pp. 72–79.

Lewis, M., Yarats, D., Dauphin, Y., Parikh, D., and Batra, D. (2017). Deal or No Deal? End-to-End Learning of Negotiation Dialogues. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP-2017)*, pp. 2443–2453.

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016a). Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP-2016)*

Li, X., Lipton, Z. C., Dhingra, B., Li, L., Gao, J., and Chen, Y.-N. (2016b). A user simulator for task-completion dialogues. ArXiv:1612.05688.

Li, J., Miller, A. H., Chopra, S., Ranzato, M., and Weston, J. (2017). Learning through dialogue interactions by asking questions. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning 8*(3-4):293–321.

Lindgren, N. (1968). Purposive systems: The edge of knowledge. *IEEE spectrum 5*(4):89–100. IEEE.

Lipton, Z. C., Li, X., Gao, J., Li, L., Ahmed, F., and Deng, L. (2018). BBQ-Networks: Efficient Exploration in Deep Reinforcement Learning for Task-Oriented Dialogue Systems. AAAI. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 5237–5244.

Lison, P. (2013). Model-based bayesian reinforcement learning for dialogue management. *INTERSPEECH*, pp. 475–479.

Litman, D., Singh, S., Kearns, M., and Walker, M. (2000). NJFun: a reinforcement learning spoken dialogue system. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems*, pp. 17–20. ACL.

Littman, M. L., Sutton, R. S., Singh. (2002). Predictive representations of state. In *Proceedings of the 14th Conference on Neural Information Processing Systems (NIPS 2001)*, pp. 1555–1561. MIT Press, Cambridge, MA.

Liu, B. (2018). *Learning Task-Oriented Dialog with Neural Network Methods*. Ph.D. thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering.

Liu, B. and Lane, I. (2017a). Iterative policy learning in end-to-end trainable task-oriented neural dialog models. In *Automatic Speech Recognition and Understanding Workshop*, pp. 482–489.

Liu, B. and Lane, I. (2017b). Multi-domain adversarial learning for slot filling in spoken language understanding. In *1st Workshop on Conversational AI, NIPS 2017*.

Liu, B., Tür, G., Hakkani-Tür, D., Shah, P., and Heck, L. (2017). End-to-end optimization of task-oriented dialogue model with deep reinforcement learning. ArXiv:1711.10712.

Liu, B., Tür, G., Hakkani-Tür, D., Shah, P., and Heck, L. (2018). Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2060–2069. New Orleans, Louisiana: ACL.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning, 8*(3-4):293–321.

Loftin, R. T., MacGlashan, J., Peng, B., Taylor, M. E., Littman, M. L., Huang, J., et al. (2014). A strategy-aware technique for learning behaviors from discrete human feedback. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-14)*, pp. 937–943.

Long, M. H. (1981). Input, interaction and second language acquisition. *Annals of the New York Academy of Sciences 379*, 259–278.

Lu, Y., and Smith, S. (2007). Augmented reality e-commerce assistant system: trying while shopping. In *International Conference on Human-Computer Interaction*, pp. 643–652. Springer, Berlin, Heidelberg.

Lucas, M., Miko, S., and Bennington, S. A. (2004). *Method and apparatus for voice dictation and document production.* US Patent 6,834,264. Google Patents.

Maedche, A., Morana, S., Schacht, S., Werth, D., and Krumeich, J. (2016). Advanced user assistance systems. *Business and Information Systems Engineering, 58*(5), 367–370.

Maheswaran, R. T., Tambe, M., Varakantham, P., and Myers, K. (2003). Adjustable autonomy challenges in personal assistant agents: A position paper. In *International Workshop on Computational Autonomy, AAMAS*, pp. 187–194. Springer.

Manuvinakurike, R., Bui, T., Chang, W., and Georgila, K. (2018). Conversational image editing: Incremental intent identification in a new dialogue task. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue.* pp. 284–295.

Mendez, J. A., Geramifard, A., Ghavamzadeh, M., and Liu, B. (2019). Reinforcement learning of multi-domain dialog policies via action embeddings. In *3rd Workshop*

*on Conversational AI, NIPS 2019*.

McDermott, J. (1982). XSEL: A computer salesperson's assistant. *Machine intelligence, 10*(1), 235–337. John Wiley and Sons.

Microsoft (2020). Windows Speech Recognition commands. https://support.microsoft.com. [Accessed February 23, 2020]

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th Conference on Neural Information Processing Systems (NIPS 2013)*, pp. 3111–3119.

Minton, S. (1988). *Learning search control knowledge. An explanation-based approach*. Kluwer Academic Publishers.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). Playing Atari with Deep Reinforcement Learning. In *Deep Learning Workshop, NIPS 2013*.

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., Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540):529–533.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd international conference on Machine learning (ICML-16)*, pp. 1928–1937. PMLR.

Moar, J., and Escherich, M. (2021). *Voice Assistant Transaction Values To Grow By Over 320% By 2023* [Press release]. Retrieved from

https://www.juniperresearch.com/researchstore/devices-technology/voice-assistants-market-research-report. Juniper Research.

Moore, A. W., Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning, 13*(1):103–130.

Moore, J. D., and Paris, C. L. (1989). Planning text for advisory dialogues. In *Proceedings of the 27th Annual Meeting on Association for Computational Linguistics*, pp. 203–211. ACL.

Mordatch, I. and Abbeel, P. (2018). Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*.

Naik, A., Shariff, R., Yasui, N., and Sutton, R. S. (2019). Discounted reinforcement learning is not an optimization problem. *Optimization Foundations of Reinforcement Learning Workshop, NIPS 2019*.

Naik, A., Abbas, Z., White, A., Sutton R. S. (2021) In *Never-Ending Reinforcement Learning (NERL) Workshop, ICLR*.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. In *Proceedings of the 30st Conference on Neural Information Processing Systems (NIPS 2016)*, eds. S. Thrun, L. K. Saul, and B. Schölkopf, pp. 799–806. MIT Press.

Nonaka, Y., Sakai, Y., Yasuda, K., and Nakano, Y. (2012). Towards assessing the communication responsiveness of people with dementia. In *International Conference on Intelligent Virtual Agents*. pp. 496–498. Springer, Berlin, Heidelberg.

Nortmann, N., Rekauzke, S., Onat, S., König, P., and Jancke, D. (2015). Primary visual cortex represents the difference between past and present. *Cerebral Cortex*

*25*, 1427–1440.

Nuance (2003). Dragon NaturallySpeaking 13 Installation Guide and User Guide. https://www.nuance.com [Accessed January 5, 2021]

Oh, J., Guo, X., Lee, H., Lewis, R. L., Singh, S. (2015). Action-conditional video prediction using deep networks in Atari games. In *Proceedings of the 29th Conference on Neural Information Processing Systems (NIPS 2015)*, pp. 2845–2853. Curran Associates, Inc.

Pan Y., Zaheer M., White A., Patterson A., and White M. (2018). Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18)*, pp. 4794–4800. AAAI Press.

Pan Y., Yao H., Farahmand A.M., White M. (2019). Hill Climbing on Value Estimates for Search-control in Dyna. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19)*.

Paolacci, G., Chandler, J., and Ipeirotis, P. G. (2010). Running Experiments on Amazon Mechanical Turk. *Judgment and Decision making 5*:411–419.

Papangelis, A., Namazifar, M., Khatri, C., Wang, Y.-C., Molino, P., and Tür, G. (2020). Plato Dialogue System: A Flexible Conversational AI Research Platform. ArXiv:2001.06463.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning (ICML-08)*, pp. 752–759.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solv-*

*ing*. Addison-Wesley, Reading, MA.

Peng, B., Li, X., Li, L., Gao, J., Celikyilmaz, A., Lee, S., et al. (2017). Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (Copenhagen, Denmark: Association for Computational Linguistics), pp. 2231–2240.

Peng, B., Li, X., Gao, J., Liu, J., Chen, Y. N., and Wong, K. F. (2018a). Adversarial advantage actor-critic model for task-completion dialogue policy learning. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6149–6153. IEEE.

Peng, B., Li, X., Gao, J., Liu, J., and Wong, K.-F. (2018b). Deep Dyna-Q: Integrating Planning for Task-Completion Dialogue Policy Learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive behavior, 1*(4): 437–454.

Pica, T. (1987). Second-language acquisition, social interaction, and the classroom. *Applied Linguistics 8*, 3–21.

Pica, T., Doughty, C. J., and Young, R. (1986). Making input comprehensible: Do interactional modifications help? *ITL-International Journal of Applied Linguistics 72*, 1–25.

Pineau, J., Montemerlo, M., Pollack, M., Roy, N., and Thrun, S. (2003). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and autonomous systems 42*:271–281.

Pilarski, P. M., Dawson, M. R., Degris, T., Fahimi, F., Carey, J. P., and Sutton, R. S. (2011). Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *Proceedings of the 2011 IEEE International Conference on Rehabilitation Robotics (ICORR)*, pp. 134–140. Zurich, Switzerland: IEEE.

Pilarski, P. M., Sutton, R. S., Mathewson, K. W., Sherstan, C., Parker, A. S., and Edwards, A. L. (2017). Communicative Capital for Prosthetic Agents. ArXiv:1711.03676.

Pollack, M. E., Brown, L., Colbry, D., Orosz, C., Peintner, B., Ramakrishnan, S., et al. (2002). Pearl: A mobile robotic assistant for the elderly. AAAI Technical Report WS-02-02. pp. 85–91. AAAI.

Pollack, M. E., Hirschberg, J., and Webber, B. (1982). User participation in the reasoning processes of expert systems. In *Proceedings of the 1st AAAI Conference on Artificial Intelligence (AAAI-82)*, pp. 358–361. University of Pennsylvania.

Power, R. (1974). *A computer model of conversation. Ph.D. thesis*, The University of Edinburg.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning.* Ph.D. thesis, University of Massachusetts, Amherst.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2015). Sequence level training with recurrent neural networks. ArXiv:1511.06732.

Quarteroni, S. and Manandhar, S. (2009). Designing an interactive open-domain question answering system. *Natural Language Engineering 15*(1):73–95.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners.* OpenAI Blog.

https://openai.com/blog/better-language-models. [Accessed July 23, 2020].

Rao, A. P. (2011). *Predictive speech-to-text input*. US Patent 7,904,298. Google Patents.

Rastogi, A., Hakkani-Tür, D., and Heck, L. (2017). Scalable multi-domain dialogue state tracking. In *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU*, pp. 561–568.

Rezende D. J., Mohamed S., Wierstra D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS 2014)*.

Saleh, A., Jaques, N., Ghandeharioun, A., Shen, J.H., and Picard, R. W. (2020). Hierarchical Reinforcement Learning for Open-Domain Dialog. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20)*, pp. 8741–8748.

Salichs, M., Ge, S., Barakova, E., Cabibihan, J., Wagner, A. R., González, Á., and He, H. (2019). Preface–Social Robotics. In *11th International Conference, ICSR-2019*. Springer.

Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., and Young, S. (2007). Agenda-Based User Simulation for Bootstrapping a POMDP Dialogue System. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 149–152.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*.

Schulman, J., Chen, X., Abbeel, P. (2017). Equivalence between policy gradients and soft Q-Learning. ArXiv:1704.06440.

Schulman J., Levine S., Abbeel P., Jordan M., and Moritz P. (2015). Trust region policy optimization. In *Proceedings of the 32nd international conference on Machine learning (ICML-15)*, pp. 1889–1897.

Schulman J., Wolski F., Dhariwal P., Radford A., and Klimov O. (2017). Proximal policy optimization algorithms. ArXiv: 1707.06347.

Schrittwieser, J., Antonoglou, I., Hubert, T. *et al.* (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature 588*, 604–609.

Selfridge, O. G. (1993). The Gardens of Learning: A Vision for AI. *AI Magazine, 14*(2):36–48.

Serban, I. V., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subramanian, S., et al. (2017a). A deep reinforcement learning chatbot. ArXiv:1709.02349.

Serban, I., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A., and Bengio, Y. (2017b). A hierarchical latent variable encoder-decoder model for generating dialogues. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*(Vol. 31, No. 1).

Shah, P., Hakkani-Tür, D., and Heck, L. (2016). Interactive reinforcement learning for task-oriented dialogue management. In *Workshop on Deep Learning for Action and Interaction, NIPS*.

Shah, P., Hakkani-Tür, D.,Tür, G., Rastogi, A., Bapna, A., Nayak, N., et al. (2018). Building a conversational agent overnight with dialogue self-play. ArXiv:1801.04871.

Shariff, R., and Szepesvári, C. (2020). Efficient Planning in Large MDPs with Weak Linear Function Approximation. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NIPS 2020)*, Vancouver, Canada.

Shen, Y., Huang, P.-S., Gao, J., and Chen, W. (2017). ReasoNet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1047–1055.

Shi, Z., Chen, X., Qiu, X., and Huang, X. (2018). Toward Diverse Text Generation with Inverse Reinforcement Learning. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18)*.

Shin, J., Xu, P., Madotto, A., and Fung, P. (2019). Happybot: Generating empathetic dialogue responses by improving user experience look-ahead. ArXiv:1906.08487.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS 2014)*, pp. 387–395.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017a). Mastering the game of Go without human knowledge. *Nature, 550*(7676):354–359.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., et al. (2017b). The Predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3191–3199.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science 362*:1140–1144.

Simmons, R., and Slocum, J. (1972). Generating English discourse from semantic networks. *Communications of the ACM 15*(10):891–905.

Singh, S. P. Reinforcement learning with a hierarchy of abstract models. (1992). In

*Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 202–207. AAAI/MIT Press, Menlo Park, CA.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In *Proceedings of the 7th Conference on Neural Information Processing Systems (NIPS 1994)*, pp. 359–368. MIT Press, Cambridge, MA.

Singh, S. P., Kearns, M. J., Litman, D. J., and Walker, M. A. (2000). Reinforcement learning for spoken dialogue systems. In *Proceedings of the 13th Conference on Neural Information Processing Systems (NIPS 2000)*, pp. 956–962.

Singh, S., Litman, D., Kearns, M., and Walker, M. (2002). Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research 16*:105–133.

Skantze, G. (2016). Real-time coordination in human-robot interaction using face and voice. *AI Magazine 37*:19–31.

Smith, R. W., and Hipp, D. R. (1994). *Spoken natural language dialog systems: A practical approach*. Oxford University Press.

Smith, E. M., Williamson, M., Shuster, K., Weston, J., and Boureau, Y.-L. (2020). Can You Put it All Together: Evaluating Conversational Agents' Ability to Blend Skills. ArXiv:2004.08449.

Stent, A., Prasad, R., and Walker, M. (2004). Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, ACL, pp. 79—87.

Sorg, J., Singh, S. Linear options. (2010). In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 31–38.

Su, P.-H., Budzianowski, P., Ultes, S., Gašić, M., and Young, S. (2017). Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management. In *Proceedings of the SIGDIAL for Dialogue Policy Learning*, pp. 147-157.

Su, S.-Y., Li, X., Gao, J., Liu, J., and Chen, Y.-N. (2018). Discriminative Deep Dyna-Q: Robust Planning for Dialogue Policy Learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP-2018)*, pp. 3813–3823. ACL.

Sukhbaatar, S., Szlam, A., and Fergus, R. (2016). Learning multiagent communication with backpropagation. In *Proceedings of the 30st Conference on Neural Information Processing Systems (NIPS 2016)*, pp. 2244–2252.

Sun, M., Chen, Y.-N., and Rudnicky, A. I. (2016). An intelligent assistant for high-level task understanding. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pp. 169–174. Sonoma, USA. ACM.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS 2014)*, pp. 3104–3112.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning, 3*(1): 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Workshop on Machine Learning (ICML-90)*, pp. 216–224. Morgan Kaufmann. Also appeared as "Artificial intelligence by dynamic programming," in *Proceedings of the*

*Sixth Yale Workshop on Adaptive and Learning Systems*, pp. 89–95.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin 2*(4):160–163. ACM, New York.

Sutton, R. S. (2019). *The Bitter Lesson* [Blog post]. Retrieved from http://www.incompleteideas.net/IncIdeas/BitterLesson.html. [Accessed June 20, 2020]

Sutton, R. S. (2020). *Experience and Intelligence: Toward a Scalable AI-Agent Architecture*. Vector Institute for Artificial Intelligence, Visitor Research Talk.

Sutton, R. S. and Barto, A. G. (1981). An adaptive network that constructs and uses and internal model of its world. *Cognition and Brain Theory 4*:217–246.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An Introduction*. MIT press.

Sutton, R. S. and Pinette, B. (1985). The learning of world models by connectionist networks. In *Proceedings of the 7th Annual Conference of the Cognitive Science Society*. pp. 54–64.

Sutton, R. S., Precup, D., Singh, S. (1998). Intra-option learning about temporally abstract actions. In *Proceedings of the 15th international conference on Machine learning (ICML-98)*, pp. 556-564. Morgan Kaufmann.

Sutton, R. S., Precup, D., Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence, 112*(1-2):181–211.

Sutton, R. S. , Rafols, E. J. , and Koop, A. (2005). Temporal Abstraction in Temporal-difference Networks. In *Proceedings of the 19th Conference on Neural Information Processing Systems (NIPS 2005)*.

Sutton, R. S., Szepesvári, Cs., Geramifard, A., Bowling, M. (2008). Dyna-style planning with linear function approximation and prioritized sweeping, In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pp. 528–536.

Takahashi, F. (2001). *Document editing system and method.* US Patent 6,202,073. Google Patents.

Talvitie, E. (2014). Model Regularization for Stable Sample Rollouts. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pp. 780–789.

Talvitie, E. (2017). Self-correcting Models for Model-based Reinforcement Learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17).*

Tang, D., Li, X., Gao, J., Wang, C., Li, L., and Jebara, T. (2018). Subgoal discovery for hierarchical dialogue policy learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP-2018)*, pp. 2298–2309.

Thomaz, A. L., Hoffman, G., and Breazeal, C. (2005). Real-time interactive reinforcement learning for robots. In *Workshop on human comprehensible machine learning, AAAI.*

Thomaz, A. L., Hoffman, G., and Breazeal, C. (2006). Reinforcement Learning with Human Teachers: Understanding How People Want to Teach Robots. In *ROMAN 2006–The 15th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 352–357.

Thompson, C. (2006). Google's China Problem. *The Power of Information.* School of Journalism, Stony Brook University.

Tractica, A market intelligence firm. (2016). The Virtual Digital Assistant Market Will Reach $15.8 Billion Worldwide by 2021. https://www.tractica.com/newsroom/

press-releases/the-virtual-digital-assistant-market-will-reach-15-8-billion-worldwide-by-2021 [Accessed January 12, 2019]

Tür, G., Hakkani-Tür, D., and Schapire, R. E. (2005). Combining active and semi-supervised learning for spoken language understanding. *Speech Communication, 45*(2):171–186.

Tür, G., *et al.* (2010). The CALO meeting assistant system. in *IEEE Transactions on Audio, Speech, and Language Processing, 18*(6), 1601–1611.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, AAAI Press.

van Hasselt, H. P., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NIPS 2019)*, pp. 14322–14333.

van Seijen, H., Nekoei, H., Racah, E., and Chandar, S. (2020). The LoCA Regret: A Consistent Metric to Evaluate Model-Based Behavior in Reinforcement Learning. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NIPS 2020)*.

Veeriah, V., Pilarski, P. M., and Sutton, R. S. (2016). Face valuing: Training user interfaces with facial expressions and reinforcement learning. In *Workshop on Interactive Machine Learning, IJCAI*.

Wahlström, N., Schön, T. B., and Deisenroth, M. P. (2015). From pixels to torques: Policy learning with deep dynamical models. *Deep Learning Workshop at the 32nd International Conference on Machine Learning, ICML*.

Walker, D. E., and Grosz, B. J. (1978). *Understanding spoken language*. Elsevier

Science Inc.

Walker, R. C. (1998). *Text processor.* US Patent 5,802,533. Google Patents.

Walker, M. A., Stent, A., Mairesse, F., and Prasad, R. (2007). Individual and domain adaptation in sentence planning for dialogue. *Journal of Artificial Intelligence Research 30*:413–456.

Wan, Y., Zaheer, M., White, A., White, M., and Sutton, R. S. (2019). Planning with expectation models. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19)*, pp.3649–3655.

Wang, Y., Si, P., Lei, Z., Xun, G., and Yang, Y. (2019). HSCJN: A Holistic Semantic Constraint Joint Network for Diverse Response Generation. In *3rd Workshop on Conversational AI, NIPS 2019.*

Waters, R. C. (1986). KBEmacs: Where's the AI?. *AI Magazine, 7*(1), 47–47.

Watkins, C. (1989).*Learning from delayed rewards.* Ph.D. thesis, King's College, Cambridge, England.

Watkins, C. J. C. H., Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3-4):279–292.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Proceedings of the 29th Conference on Neural Information Processing Systems (NIPS 2015)*, pp. 2746–2754. Cambridge,USA: MIT Press.

Weisz, G., Budzianowski, P., Su, P.-H., and Gašić, M. (2018). Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *IEEE/ACM Transactions on Audio, Speech, and Language Processing 26*, pp. 2083–2097.

Wen, T.-H., Vandyke, D., Mrkšic, N., Gašić, M., Rojas Barahona, L. M., Su, P.-H., et al. (2017). A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pp. 437–449. Association for Computational Linguistics.

Wiering, M., Salustowicz, R., and Schmidhuber, J. (2001). Model-based reinforcement learning for evolving soccer strategies. In *Computational intelligence in games*, pp. 99–132. Springer.

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

Williams, J. D., Asadi, K., and Zweig, G. (2017). Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. ACL.

Winograd, T. (1971). Procedures as a representation for data in a computer program for understanding natural language. Technical Report, MIT.

Witten, I. H. (1977). An adaptive optimal controller for discrete-time Markov environments. *Information and Control, 34*(4):286–295.

White A. (2015). *Developing a predictive approach to knowledge*. Ph.D. thesis, University of Alberta.

Woodrow, H. (1946). The ability to learn. *Psychological Review 53*, 147–158.

Woods, W. (1984). Natural language communication with machines: An ongoing goal. *Artificial intelligence applications for business*, pp. 195–209.

Wu, Y., Li, X., Liu, J., Gao, J., and Yang, Y. (2019). Switch-based active deep Dyna-Q: Efficient adaptive planning for task-completion dialogue policy learning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-19)*, pp. 7289–7296.

Xu, C., Wu, W., and Wu, Y. (2018). Towards explainable and controllable open domain dialogue generation with dialogue acts. ArXiv:1807.07255.

Yao, H., Bhatnagar, S., and Diao, D. (2009). Multi-step linear Dyna-style planning. In *Proceedings of the 23rd Conference on Neural Information Processing Systems (NIPS 2009)*, pp. 2187–2195.

Yarats, D. and Lewis, M. (2017). Hierarchical Text Generation and Planning for Strategic Dialogue. In *Proceedings of the 34th international conference on Machine learning (ICML-17)*. Stockholm, Sweden: PMRL 80.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*.

Zhao, T., and Eskenazi, M. (2016). Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In *Proceedings of the SIGDIAL 2016 Conference*, p. 1–10. Los Angeles: ACL.

Zhao, Y., Wang, Z., Yin, K., Zhang, R., Huang, Z., and Wang, P. (2020). Dynamic Reward-Based Dueling Deep Dyna-Q: Robust Policy Learning in Noisy Environments. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20)*, pp. 9676–9684.

Zhang, J., Zhao, T., and Yu, Z. (2018). Multimodal hierarchical reinforcement learning policy for task-oriented visual dialog. *Proceedings of the 19th Annual SIGdial*

*Meeting on Discourse and Dialogue*.

Zhang, Z., Takanobu, R., Zhu, Q., Huang, M., and Zhu, X. (2020). Recent advances and challenges in task-oriented dialog systems. *Science China Technological Sciences*, pp. 1–17.

Zhou, L., Small, K., Rokhlenko, O., and Elkan, C. (2017). End-to-end offline goal-oriented dialog policy learning via policy gradient. In *1st Workshop on Conversational AI, NIPS 2017*.

Zhou, L., Gao, J., Li, D., and Shum, H. Y. (2020). The design and implementation of XiaoIce, an Empathetic Social Chatbot. *Computational Linguistics, 46*(1):53–93.

Zhou, M., Arnold, J., and Yu, Z. (2019). Building task-oriented visual dialog systems through alternative optimization between dialog policy and language generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27.

# Appendix A

# Model Update

Here, we describe a general update for an expectation model. This update is applicable for our settings of function approximation and linear value functions.

Recall that $\mathbf{s} \in \mathbb{R}^d$ is a feature vector of an agent state. The model consists of a forward transition matrix $\mathcal{F} \in \mathbb{R}^d \times \mathbb{R}^d$ and an expected reward vector $\boldsymbol{\eta} \in \mathbb{R}^d$ constructed such that $\hat{\mathbf{s}}(\mathbf{s}, a, \boldsymbol{\eta}) \doteq \mathcal{F}\mathbf{s}$ and $\hat{r}(\mathbf{s}, a, \boldsymbol{\eta}) \doteq \boldsymbol{\eta}^\top \mathbf{s}$ can be used as estimates of the feature vector and reward that follow the state $\mathbf{s}_t$. We omit the time step $t$ to simplify a notation in the update and write $\mathbf{s}$. Thus, at a planning step, the outputs of the model are a next feature vector $\hat{\mathbf{s}}(\mathbf{s}, a, \boldsymbol{\eta}) \doteq \mathcal{F}\mathbf{s}$ and next reward $\hat{r}(\mathbf{s}, a, \boldsymbol{\eta}) \doteq \boldsymbol{\eta}^\top \mathbf{s}$.

Here, we use $\hat{v}$ as an arbitrary linear value function $\hat{v} \doteq \mathbf{w}^\top \mathbf{s}$, which can be a state-value function $\hat{v}(\mathbf{s}, \mathbf{w})$ or an action-value function $\hat{q}(\mathbf{s}, a, \mathbf{w})$ parameterized by learnable weights $\mathbf{w} \in \mathbb{R}^d$.

During the model update we update a matrix $\mathcal{F}$ and a reward vector $\boldsymbol{\eta}$.

### A.0.1 Matrix Update

To update the matrix $\mathcal{F}$, we want to minimize mean square error (MSE) between the next state feature vector $\hat{s}$ that is an output of the model and the next feature vector $\mathbf{s}'$ the agent will receive. The MSE is computed as L2 norm between these two vectors:

$$\text{MSE} \doteq ||\hat{\mathbf{s}}(\mathbf{s}, a, \boldsymbol{\eta}) - (\mathbf{s}')||^2 = ||\mathcal{F}\mathbf{s} - (\mathbf{s}')||^2 = (\mathcal{F}\mathbf{s})^\top \mathcal{F}\mathbf{s} - 2(\mathbf{s}')^\top \mathcal{F}\mathbf{s} + (\mathbf{s}')^\top (\mathbf{s}')$$

We use a classic stochastic gradient descent (SGD) method to adjust the matrix. Thus, we want to take a gradient of MSE with respect to $\mathcal{F}$:

$$\frac{\nabla(\text{MSE})}{\nabla_{\mathcal{F}}} = \nabla\Big[(\mathcal{F}\mathbf{s})^\top \mathcal{F}\mathbf{s} - 2(\mathbf{s}')^\top \mathcal{F}\mathbf{s} + (\mathbf{s}')^\top \mathbf{s}\Big] = \nabla\Big[(\mathcal{F}\mathbf{s})^\top \mathcal{F}\mathbf{s} - 2(\mathbf{s}')^\top \mathcal{F}\mathbf{s}\Big] \tag{A.1}$$

The gradient of a scalar with respect to a matrix tells us how to change the component of the matrix to get closer to that value. Consider now a partial derivative $\frac{\partial(\mathcal{F}\mathbf{s})^\top \mathcal{F}\mathbf{s}}{\partial \mathcal{F}}$. We ask how does this change if we change the $pq$-th component of the matrix where $p$ are rows and $q$ are columns of $\mathcal{F}$. We know that the 2-norm of the vector is the sum of its $i$th components:

$$\frac{\partial(\mathcal{F}\mathbf{s})^\top \mathcal{F}\mathbf{s}}{\partial \mathcal{F}_{pq}} = \frac{\partial}{\partial \mathcal{F}_{pq}}\Big[\sum_i \big(\mathcal{F}\mathbf{s}\big)_i^2\Big].$$

Now we can move the gradient inside of the sum and take the derivative:

$$\frac{\partial}{\partial \mathcal{F}_{pq}}\Big[\sum_i \big(\mathcal{F}\mathbf{s}\big)_i^2\Big] = \sum_i \frac{\partial}{\partial \mathcal{F}_{pq}}\big(\mathcal{F}\mathbf{s}\big)_i^2 = \sum_i 2\big(\mathcal{F}\mathbf{s}\big)_i \frac{\partial}{\partial \mathcal{F}_{pq}}\big(\mathcal{F}\mathbf{s}\big)_i \tag{A.2}$$

Consider now the last term in (A.2):

$$\frac{\partial}{\partial \mathcal{F}_{pq}}\big(\mathcal{F}\mathbf{s}\big)_i = \frac{\partial\big(\mathcal{F}\mathbf{s}\big)_i}{\partial \mathcal{F}_{pq}}$$
$$= \frac{\partial}{\partial \mathcal{F}_{pq}} \sum_j \mathcal{F}_{ij}\mathbf{s}_j = \begin{cases} 0, & \text{if } i \neq p, \text{ the } \mathcal{F}_{pq} \text{ does not show up in the sum} \\ \mathbf{s}_q, & \text{if } i = p \end{cases}$$

We know that $\mathbf{s}_q = \mathbf{s}^\top$, thus, based on (A.2) and we can write

$$\frac{\nabla(\mathcal{F}\mathbf{s})^\top \mathcal{F}\mathbf{s}}{\nabla \mathcal{F}} = 2(\mathcal{F}\mathbf{s})\mathbf{s}^\top.$$

Consider now a second partial derivative $\frac{\partial(-2(\mathbf{s}')^\top \mathcal{F}\mathbf{s})}{\partial \mathcal{F}}$, the shape of which is a matrix. We again ask how does this change if we change the $pq$-th component of the matrix where $p$ are rows and $q$ are columns of $\mathcal{F}$. By matrix multiplication formula

$$\frac{\partial(-2(\mathbf{s}')^\top \mathcal{F}\mathbf{s})}{\partial \mathcal{F}_{pq}} = \frac{\partial}{\partial \mathcal{F}_{pq}}\bigg[ -2 \sum_{ij} \mathcal{F}_{ij}(\mathbf{s}')_i \mathbf{s}_j \bigg].$$

Then the derivative of this will only be non-zero when the index $ij = pq$, and for all other terms it will disappear:

$$\frac{\partial}{\partial \mathcal{F}_{pq}}\bigg[ -2 \sum_{ij} \mathcal{F}_{ij}(\mathbf{s}')_i \mathbf{s}_j \bigg] = -2(\mathbf{s}')_p \mathbf{s}_q.$$

Now we can define a matrix which is a rank 1 matrix:

$$\frac{\nabla(-2\mathbf{s}^\top \mathcal{F}\mathbf{s})}{\nabla \mathcal{F}} = -2 \begin{bmatrix} (\mathbf{s}')_1\mathbf{s}_1 & \dots & (\mathbf{s}')_1\mathbf{s}_d \\ \dots & \dots & \dots \\ (\mathbf{s}')_d\mathbf{s}_1 & \dots & (\mathbf{s}')_d\mathbf{s}_d \end{bmatrix} = -2(\mathbf{s}')\mathbf{s}^\top.$$

Now we write all the terms for (A.1):

$$\nabla\Big[(\mathcal{F}\mathbf{s})^\top\mathcal{F}\mathbf{s} - 2\mathbf{s}^\top\mathcal{F}\mathbf{s}\Big] = 2(\mathcal{F}\mathbf{s})\mathbf{s}^\top - 2(\mathbf{s}')\mathbf{s}^\top = 2\Big[\mathcal{F}\mathbf{s} - (\mathbf{s}')\Big]\mathbf{s}^\top.$$

Then the model update with a step size $\alpha$ is

$$\begin{aligned}\mathcal{F}_{t+1} &\doteq \mathcal{F}_t - \alpha\Big[\mathbf{s}_{t+1} - \mathcal{F}\mathbf{s}_t\Big]\mathbf{s}_t^\top \\ &= \mathcal{F}_t + \alpha\Big[\mathcal{F}\mathbf{s}_t - \mathbf{s}_{t+1}\Big]\mathbf{s}_t^\top. \end{aligned} \tag{A.3}$$

### A.0.2 Reward Vector Update

To update a reward vector $\boldsymbol{\eta}$ with a step size $\alpha$

$$\begin{aligned}\boldsymbol{\eta}_{t+1} &\doteq \boldsymbol{\eta}_t - \alpha[\hat{r}(\mathbf{s}_t, a, \boldsymbol{\eta}_t) - r]\nabla\hat{r}(\mathbf{s}_t, a, \boldsymbol{\eta}_t) \\ &= \boldsymbol{\eta}_t - \alpha[\hat{r}(\mathbf{s}_t, a, \boldsymbol{\eta}_t) - r]\nabla\boldsymbol{\eta}_t^\top\mathbf{s}_t \\ &= \boldsymbol{\eta}_t + \alpha[r - \boldsymbol{\eta}_t^\top\mathbf{s}_t]\mathbf{s}_t. \end{aligned} \tag{A.4}$$

# Glossary

**A3C** Asynchronous Advantage Actor-Critic.

**AC** Actor-Critic.

**Adjunct $\pi$** Adjunct Policy.

**Adjunct Q** Adjunct Action-value Function.

**DQN** Deep Q-networks.

**DTP** Decide-Time Planning.

**ER** Experience Replay.

**GRU** Gated Recurrent Units.

**MBPO** Model-Based Policy Optimization.

**MDP** Markov Decision Process.

**MSE** Mean Squared Error.

**NN** Neural Networks.

**POMDP** Partially Observable Markov Decision Process.

**RL** Reinforcement Learning.

**RNN** Recurrent Neural Network.

**SGD** Stochastic Gradient Descent.

**SPPO** Soft-Planner Policy Optimization.

**SVM** Support Vector Machine.

**TD** Temporal Difference.

**VDE** Voice Document Editing.

# Index