Journal of Artificial Intelligence Research 69 (2020) 571-612 Submitted 07/2018; published 10/2020

The Bottleneck Simulator: A Model-Based Deep Reinforcement Learning Approach

Iulian Vlad Serban IULIAN.VLAD.SERBAN@UMONTREAL.CA
Chinnadhurai Sankar CHINNADHURAI.SANKAR@UMONTREAL.CA
Mila (Quebec Artificial Intelligence Institute)
Department of Computer Science and Operations Research
University of Montreal, Montreal, Canada

Michael Pieper MPIEPER636@GMAIL.COM
Polytechnique Montreal
Montreal, Canada

Joelle Pineau JPINEAU@CS.MCGILL.CA
Mila (Quebec Artificial Intelligence Institute)
School of Computer Science, McGill University
Montreal, Canada

Yoshua Bengio YOSHUA.BENGIO@MILA.QUEBEC
Mila (Quebec Artificial Intelligence Institute)
Department of Computer Science and Operations Research
University of Montreal, Montreal, Canada

Abstract

Deep reinforcement learning has recently shown many impressive successes. However, one major obstacle towards applying such methods to real-world problems is their lack of data-efficiency. To this end, we propose the Bottleneck Simulator: a model-based reinforcement learning method which combines a learned, factorized transition model of the environment with rollout simulations to learn an effective policy from few examples. The learned transition model employs an abstract, discrete (bottleneck) state, which increases sample efficiency by reducing the number of model parameters and by exploiting structural properties of the environment. We provide a mathematical analysis of the Bottleneck Simulator in terms of fixed points of the learned policy, which reveals how performance is affected by four distinct sources of error: an error related to the abstract space structure, an error related to the transition model estimation variance, an error related to the transition model estimation bias, and an error related to the transition model class bias. Finally, we evaluate the Bottleneck Simulator on two natural language processing tasks: a text adventure game and a real-world, complex dialogue response selection task. On both tasks, the Bottleneck Simulator yields excellent performance beating competing approaches.

1. Introduction

Deep reinforcement learning (RL) has recently shown impressive successes across a variety of tasks (Mnih et al., 2013; Tesauro, 1995; Silver et al., 2017, 2016; Brown & Sandholm, 2017; Watter et al., 2015; Lillicrap et al., 2016; Schulman et al., 2015; Levine et al., 2016). However, the silver bullet for many of these successes have been enormous amounts of training data and result in policies which do not generalize to changes or novel tasks in the environment. Fewer

successes have been achieved outside the realm of simulated environments or environments where agents can play against themselves. Thus, one major impediment towards applying deep RL to real-world problems is a lack of data-efficiency.

One promising solution is *model-based* RL, where an internal model of the environment is learned. By learning an internal environment model the agent may be able to exploit structural properties of the environment. This enables the agent to reduce the amount of trial-and-error learning and to better generalize across states and actions.

In this paper we propose a model-based RL method based on learning an approximate, factorized transition model. The approximate transition model involves discrete, abstract states acting as information bottlenecks, which mediate the transitions between successive full states. Once learned, the approximate transition model is then applied to learn the agent's policy (for example, using Q-learning with rollout simulations). This method has several advantages. First, the factorized model has significantly fewer parameters compared to a non-factorized transition model, making it highly sample efficient. Second, by learning the abstract state representation with the specific goal of obtaining an optimal policy (as opposed to maximizing the transition model's predictive accuracy), it may be possible to trade-off some of the transition model's predictive power for an improvement in the policy's performance. By grouping similar states together into the same discrete, abstract state, it may be possible to improve the performance of the policy learned with the approximate transition model.

The idea of grouping similar states together has been proposed before in a variety of forms, such as state aggregation (Bean, Birge, & Smith, 1987; Bertsekas & Castanon, 1989; Dietterich, 2000; Jong & Stone, 2005; Jiang, Kulesza, & Singh, 2015). In contrast to many previous approaches, in our method the grouping is applied exclusively within the approximate transition model, while the agent's policy still operates on the complete world state. This allows the agent's policy (e.g. a neural network) to form its own high-level, distributed representations of the world from the complete world state. Importantly, in this method, the agent's policy is capable of obtaining better performance compared to standard state aggregation, because it may counter deficiencies in the abstract state representation by optimizing for myopic (next-step) rewards which it can do efficiently by accessing the complete world state. This is particularly advantageous when it is possible to pretrain the policy to imitate a myopic human policy (e.g. by imitating single actions or preferences given by humans) or with a policy learned on a similar task. Furthermore, as with state aggregation, the grouping may incorporate prior structural knowledge. As shown by the experiments, by leveraging simple knowledge of the problem domain significant performance improvements are obtained.

Our contributions are three-fold. First, we propose a model-based RL method, called the Bottleneck Simulator, which learns an approximate transition distribution with discrete abstract states acting as information bottlenecks. We formally define the Bottleneck Simulator and its corresponding Markov decision process (MDP) and describe the training algorithm in details. Second, we provide a mathematical analysis based on fixed points. We provide two upper bounds on the error incurred when learning a policy with an approximate transition distribution: one for general approximate transition distributions and one for the Bottleneck Simulator. In particular, the second bound illustrates how the overall error may be attributed to four distinct sources: an error related to the abstract space structure

(structural discrepancy), an error related to the transition model estimation variance, an error related to the transition model estimation bias, and an error related to the transition model class bias. Finally, we demonstrate the data-efficiency of the Bottleneck Simulator on two tasks involving few data examples: a text adventure game and a real-world, complex dialogue response selection task. We demonstrate how efficient abstractions may be constructed and show that the Bottleneck Simulator beats competing methods on both tasks. Finally, we investigate the learned policies qualitatively and, for the text adventure game, measure how performance changes as a function of the learned abstraction structure.

2. Background

This section provides a brief technical background.

2.1 Definitions

A Markov decision process (MDP) is a tuple $\langle S, A, P, R, \gamma \rangle$, where S is the set of states, A is the set of actions, P is the state transition probability function, $R(s,a) \in [0, r_{\max}]$ is the reward function, with $r_{\max} > 0$, and $\gamma \in (0,1)$ is the discount factor (Sutton & Barto, 1998). We adopt the standard MDP formulation with finite horizon. At time t, the agent is in a state $s_t \in S$, takes an action $a_t \in A$, receives a reward $r_t = R(s_t, a_t)$ and transitions to a new state $s_{t+1} \in S$ with probability $P(s_{t+1}|s_t, a_t)$.

We assume the agent follows a stochastic policy π. Given a state $s \in S$, the policy π assigns a probability to each possible action $a \in A$: $\pi(a|s) \in [0,1]$, s. t. $\sum_{a \in A} \pi(a|s) = 1$. The agent's goal is to learn a policy maximizing the discounted sum of rewards: $R = \sum_{t=1}^{T} \gamma^t r_t$, called the *cumulative return.* or, more briefly, the *return.*

Given a policy π, the *state-value function* V^π is defined as the expected return of the policy starting in state $s \in S$:

$$V^\pi(s) = \mathrm{E}_\pi \left[\sum_{t=1}^{T} \gamma^t r_t \mid s_1 = s \right]. \tag{1}$$

The *state-action-value function* Q^π is the expected return of taking action a in state s, and then following policy π:

$$Q^\pi(s,a) = \mathrm{E}_\pi \left[\sum_{t=1}^{T} \gamma^t r_t \mid s_1 = s, a_1 = a \right]. \tag{2}$$

An *optimal policy* π^* is a policy satisfying $\forall s \in S, a \in A$:

$$V^{\pi^*}(s) = V^*(s) = \max_\pi V^\pi(s). \tag{3}$$

The optimal policy can be found via dynamic programming using the Bellman optimality equations (Bertsekas & Tsitsiklis, 1995; Sutton & Barto, 1998), $\forall s \in S, a \in A$:

$$V^*(s) = \max_{a \in A} Q^*(s,a), \tag{4}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \tag{5}$$

which hold if and only if eq. (3) is satisfied. Popular algorithms for finding an optimal policy include Q-learning, SARSA and REINFORCE (Sutton & Barto, 1998).

2.2 Model-Based RL with Approximate Transition Models

Suppose we aim to learn an efficient policy for the MDP $\langle S, A, P, R, \gamma \rangle$, but without having access to the transition distribution P. However, suppose that we still have access to the set of states and actions, the discount factor γ and the (immediate) reward function for each state-action pair $R(s, a)$. This is a plausible setting for many real-world applications, including natural language processing, health care and robotics.

Suppose that a dataset $D = \{s^i, a^i, s'^i\}_{i=1}$ with $|D|$ tuples has been collected with a policy π_D acting in the true (ground) MDP (Sutton, 1990; Moore & Atkeson, 1993; Peng & Williams, 1993).[1] We can use the dataset D to estimate an approximate transition distribution P_{Approx}:

$$P_{\text{Approx}}(s'|s, a) \approx P(s'|s, a) \quad \forall s, s' \in S, a \in A. \tag{6}$$

Given P_{Approx}, we can form an approximate MDP $\langle S, A, P_{\text{Approx}}, R, \gamma \rangle$ and learn a policy π satisfying the Bellman equations in the approximate MDP, $\forall s \in S, a \in A$:

$$\begin{aligned} V_{\text{Approx}}(s) &= \max_{a \in A} Q_{\text{Approx}}(s, a), \\ Q_{\text{Approx}}(s, a) &= R(s, a) + \gamma \sum_{s' \in S} P_{\text{Approx}}(s'|s, a) V_{\text{Approx}}(s'), \end{aligned} \tag{7}$$

in the hope that $P_{\text{Approx}}(s'|s, a) \approx P(s'|s, a)$ implies $Q_{\text{Approx}}(s, a) \approx Q(s', a')$ $\forall s \in S, a \in A$ for policy π.

The most common approach is to learn P_{Approx} by counting co-occurrences in D (Moore & Atkeson, 1993):

$$P_{\text{Approx}}(s'|s, a) = \frac{\text{Count}(s, a, s')}{\text{Count}(s, a, \cdot)}, \tag{8}$$

where $\text{Count}(s, a, s')$ is the observation count for (s, a, s') and the variable $\text{Count}(s, a, \cdot)$ $= \sum_{s'} \text{Count}(s, a, s')$ is the observation count for (s, a) followed by any other state. Unfortunately, this approximation is not sample efficient, because its sample complexity for accurately estimating the transition probabilities may grow in the order of $O(|S|^2|A|)$ (see appendix).

The next section presents the Bottleneck Simulator, which learns a more sample efficient model and implements an inductive bias by utilizing information bottlenecks

3. Bottleneck Simulator

This section introduces the Bottleneck Simulator.

1. Since the reward function $R(s, a)$ is assumed known and deterministic, the dataset does not need to contain the rewards.

3.1 Definition

The Bottleneck Simulator is given by the tuple $\langle Z, S, A, P_{\text{Abs}}, R, \gamma \rangle$, where Z is a discrete set of *abstract states*, S is the set of (full) states, A is the set of actions and P_{Abs} is a set of distributions.[2] Further, we in general assume that $|Z| << |S|$.

The Bottleneck Simulator is illustrated in Figure 1. Conditioned on an abstract state $z \in Z$, a state $s \in S$ is sampled. Conditioned on a state s and an action $a \in A$, a reward r_t is outputted. Finally, conditioned on a state s and an action a, the next abstract state $z' \in Z$ is sampled. Formally, the following distributions are defined:

$$P_{\text{Abs}}(z_0) \qquad \text{Initial distribution of } z \tag{9}$$

$$P_{\text{Abs}}(z_{t+1}|s_t, a_t) \qquad \text{Transition distribution of } z \tag{10}$$

$$P_{\text{Abs}}(s_t|z_t) \qquad \text{Conditional distribution of } s \tag{11}$$

When viewed as a Markov chain, the abstract state z is a *Markovian* state: given a sequence $(z_1, s_1, a_1, \ldots, z_{t-1}, s_{t-1}, a_{t-1}, z_t)$, all future variables depend only on z_t. As such, the abstract state acts as an information bottleneck, since it has a much lower cardinality than the full states (i.e. $|Z| << |S|$). This bottleneck helps reduce sparsity and improve generalization. Furthermore, the representation for z can be learned using unsupervised learning or supervised learning on another task. It may also incorporate domain-specific knowledge.

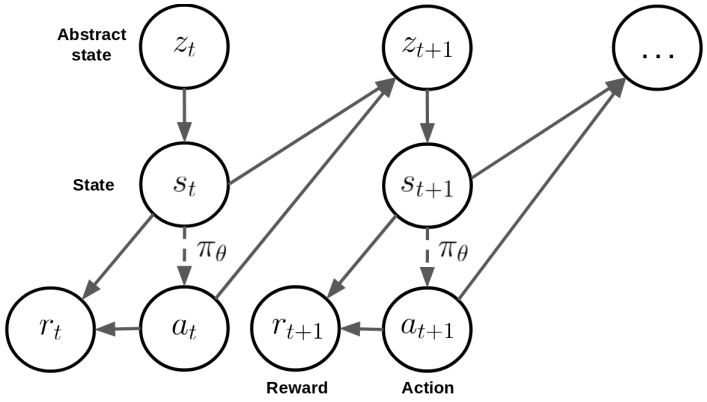

Figure 1: Probabilistic directed graphical model for the *Bottleneck Simulator*. For each time step t, z_t is a discrete random variable which represents the abstract state mediating the transitions between the successive full states s_t, a_t represents the action taken by the agent, and r_t represents the sampled reward.

Further, assume that for each state $s \in S$ there exists exactly one abstract state $z \in Z$ where $P_{\text{Abs}}(s|z)$ assigns non-zero probability. Formally, let $f_{s \to z}(s)$ be a known surjective function mapping from S to Z, such that:

$$P_{\text{Abs}}(s|z) = 0 \quad \forall s \in S, z \in Z \text{ if } \; f_{s \to z}(s) \neq z. \tag{12}$$

2. In the POMDP literature, z often represents the observation. However, in our notation, z represents the discrete, abstract state.

This assumption allows us to construct a simple estimator for the transition distribution based on $f_{s\to z}(s)$. Given a dataset $D = \{s^i, a^i, s'^i\}_{i=1}$ of tuples collected under a policy π_D acting in the true MDP, we may estimate P_{Abs} as:

$$P_{\text{Abs}}(z_0) = 1_{(f_{s\to z}(s_{\text{start}})=z_0)} \tag{13}$$

$$P_{\text{Abs}}(z_{t+1}|s_t, a_t) = \frac{\sum_{s';f_{s\to z}(s')=z_{t+1}} \text{Count}(s, a, s')}{\sum_{s'} \text{Count}(s, a, s')} \tag{14}$$

$$P_{\text{Abs}}(s_t|z_t) = \frac{\sum_{s,a} \text{Count}(s, a, s_t)}{\sum_{s,a,s';f_{s\to z}(s')=z_t} \text{Count}(s, a, s')} \tag{15}$$

This approximation has a sample complexity in the order of $O(|S||Z||A|)$ (see appendix). This should be compared to the estimator discussed previously, based on counting co-occurrences, which had a sample complexity of $O(|S|^2|A|)$. For $|Z| << |S|$, clearly $|S||Z||A| << |S|^2|A|$. As such this estimator is likely to achieve lower variance transition probability estimates for problems with large state spaces.

However, the lower variance comes at the cost of an increased bias. By partitioning the states $s \in S$ into groups, the abstract states $z \in Z$ must contain all salient information required to estimate the true transition probabilities:

$$P_{\text{Abs}}(s_{t+1}|s_t, a_t) = \sum_{z_{t+1}} P_{\text{Abs}}(z_{t+1}|s_t, a_t) P_{\text{Abs}}(s_{t+1}|z_{t+1}) \approx P(s_{t+1}|s_t, a_t). \tag{16}$$

If the abstract states cannot sufficiently capture this information, then the approximation w.r.t. the true transition distribution will be poor. This in turn is likely to cause the policy learned in the Bottleneck Simulator to yield poor performance. The same drawback applies to common state aggregation methods, such as state aggregation (Bean et al., 1987; Bertsekas & Castanon, 1989). However, unlike aggregation method, the policy learned with the Bottleneck Simulator has access to the complete, world state. Finally, it should be noted that the count-based model for P_{Abs} is still rather naïve and inefficient. In the next sub-section, we propose a more efficient method for learning P_{Abs}.

3.2 Learning

We assume that $f_{s\to z}$ is known. The transition distributions can be learned using a parametric classification model (e.g. a neural network with softmax output) by optimizing its parameters w.r.t. log-likelihood. Denote by $P_{\text{Abs}} = P_{\text{Abs},\phi}$ the transition distribution of the Bottleneck Simulator parametrized by a vector of parameters ϕ. Formally, we aim to optimize:

$$\arg\max_{\phi} \sum_{(s_t,a_t,s_{t+1},\cdot)\in D} \log P_{\text{Abs}}(s_{t+1}|s_t, a_t) \tag{17}$$

$$= \arg\max_{\phi} \sum_{(s_t,a_t,s_{t+1},\cdot)\in D} \log P_{\text{Abs}}(f_{s\to z}(s_{t+1})|s_t, a_t) + \log P_{\text{Abs}}(s_{t+1}|f_{s\to z}(s_{t+1})) \tag{18}$$

This breaks the learning problem down into two optimization problems, which are solved separately. In the appendix we propose a method for learning $f_{s\to z}$ and P_{Abs} jointly.

4. Mathematical Analysis

In this section we develop two upper bounds related to the estimation error of the state-action-value function learned in an approximate MDP. The first bound pertains to a general class of approximate MDPs and illustrates the relationship between the learned state-action-value function and the accuracy of the approximate transition distribution. The second bound relies on the hierarchical latent structure and applies specifically to the Bottleneck Simulator. This bound illustrates how the Bottleneck Simulator may learn a better policy by trading-off between four separate factors.

Define the *true* MDP as a tuple $\langle S, A, P, R, \gamma \rangle$, where S is the set of states, A is the set of actions, P is the *true* state transition distribution, $R(h, a) \in [0, r_{\max}]$ is the *true* reward function and $\gamma \in (0, 1)$ is the discount factor.

Let the tuple $\langle S, A, P_{\text{Approx}}, R, \gamma \rangle$ be an approximate MDP, where P_{Approx} is the transition function. All other variables are the same as given in the *true* MDP. Let Q_{Approx} satisfy eq. (7). This approximate MDP will serve as a reference for comparison.

Let the tuple $\langle Z, S, A, P_{\text{Abs}}, R, \gamma \rangle$ be the Bottleneck Simulator, where Z is the discrete set of abstract states and P_{Abs} is the transition function, as defined in the previous section. All other variables are the same as given in the *true* MDP. Finally, let Q_{Abs} be the optimal state-action-value function w.r.t. the Bottleneck Simulator:

$$Q_{\text{Abs}}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P_{\text{Abs}}(s'|s, a) V_{\text{Abs}}(s') \tag{19}$$

$$= R(s, a) + \gamma \sum_{s' \in S, z' \in Z} P_{\text{Abs}}(s'|z') P_{\text{Abs}}(z'|s, a) V_{\text{Abs}}(s') \tag{20}$$

We derive bounds on the loss defined in terms of distance between Q^* and sub-optimal fixed points Q_{Approx} and Q_{Abs}:

$$||Q^*(s, a) - Q_{\text{Approx}}(s, a)||_\infty, \tag{21}$$

$$||Q^*(s, a) - Q_{\text{Abs}}(s, a)||_\infty, \tag{22}$$

where $|| \cdot ||_\infty$ is the infinity norm (max norm). In other words, we bound the maximum absolute difference between the estimated return for any tuple (s, a) between the approximate state-action value and the state-action value of the optimal policy. The same loss criteria was proposed by Bertsekas and Tsitsiklis (1995, Chapter 6) as well as others.

Our first theorem bounds the loss w.r.t. an approximate MDP using either the total variation distance or the Kullback-Leibler divergence (KL-divergence). This theorem follows as a simple extension of existing results in the literature (Ross, 2013; Talvitie, 2015).

Theorem 1. *Let Q_{Approx} be the optimal state-action-value function w.r.t. the approximate MDP $\langle S, A, P_{Approx}, R, \gamma \rangle$, and let Q^* be the optimal state-action-value function w.r.t. the true MDP $\langle S, A, P, R, \gamma \rangle$. Let γ be their contraction rates. Then it holds that:*

$$||Q^*(s, a) - Q_{Approx}(s, a)||_\infty \leq \frac{\gamma r_{max}}{(1-\gamma)^2} \left|\left| \sum_{s'} \left| P(s'|s, a) - P_{Approx}(s'|s, a) \right| \right|\right|_\infty \tag{23}$$

$$\leq \frac{\gamma r_{max} \sqrt{2}}{(1-\gamma)^2} \left|\left| \sqrt{D_{KL}(P(s'|s, a)||P_{Approx}(s'|s, a))} \right|\right|_\infty, \tag{24}$$

where $D_{KL}(P(s'|s,a)||P_{Approx}(s'|s,a))$ *is the conditional KL-divergence between* $P(s'|s,a)$ *and* $P_{Approx}(s'|s,a)$.

Proof. See appendix. □

Eqs. (23) and (24) provide general bounds for any approximate transition distribution P_{Approx} (including P_{Abs}). The bounds are asymptotically tight in the sense that when P_{Approx} converges to P both the bounds go to zero. Finally, the looser bound in eq. (24) motivates why the approximate transition distribution might be learned using cross-entropy loss (or, equivalently, maximum log-likelihood).

Our second theorem bounds the loss specifically w.r.t. the Bottleneck Simulator, under the condition that if two states $s, s' \in S$ belong to the same abstract state $z \in Z$ (i.e. $f_{s\to z}(s) = f_{s\to z}(s')$) then their state-value functions are close to each other w.r.t. the optimal policy π^*: $|V^*(s) - V^*(s')| < \epsilon$ for some $\epsilon > 0$ if $f_{s\to z}(s) = f_{s\to z}(s')$. This state-value similarity is closely related to metrics based on the notion of bisimulation (Dean, Givan, & Leach, 1997; Ferns, Panangaden, & Precup, 2004; Ferns, Castro, Precup, & Panangaden, 2006; Abel, Hershkowitz, & Littman, 2016). The theorem is related to the results obtained by Ross (2013, p. 257) and Ross and Bagnell (2012), though their assumptions are different which in turn yields a bound in terms of expectations.

Theorem 2. *Let* Q_{Abs} *be the optimal state-action-value function w.r.t. the Bottleneck Simulator* $\langle Z, S, A, P_{Abs}, R, \gamma\rangle$*, and let* Q^* *be the optimal state-action-value function w.r.t. the true MDP* $\langle S, A, P, R, \gamma\rangle$*. Let* γ *be their contraction rates, and define:*

$$\epsilon = \max_{s_i, s_j \in S;\ f_{s\to z}(s_i) = f_{s\to z}(s_j)} |V^*(s_i) - V^*(s_j)| \tag{25}$$

Then it holds that:

$$\begin{aligned} ||Q^*(s,a) - Q_{Abs}(s,a)||_\infty < \frac{2\gamma\epsilon}{1-\gamma} \\ + \frac{\gamma}{1-\gamma}\left|\left|\sum_{s'\in S} V_{min}(s')\left|P_{Abs}(s'|s,a) - P^{\infty}_{Abs}(s'|s,a)\right|\right|\right|_\infty \\ + \frac{\gamma}{1-\gamma}\left|\left|\sum_{s'\in S} V_{min}(s')\left|P^{\infty}_{Abs}(s'|s,a) - P^{*}_{Abs}(s'|s,a)\right|\right|\right|_\infty \\ + \frac{\gamma}{1-\gamma}\left|\left|\sum_{s'\in S} V_{min}(s')\left|P^{*}_{Abs}(s'|s,a) - P(s'|s,a)\right|\right|\right|_\infty \end{aligned} \tag{26}$$

where V_{min} and P^{∞}_{Abs} are defined as:

$$V_{min}(s) = \min_{\substack{s' \in S, \\ f_{s\to z}(s') = f_{s\to z}(s)}} V^*(s'), \tag{27}$$

$$P^{\infty}_{Abs}(s'|s,a) = \sum_{z' \in Z} P^{\infty}_{Abs}(z'|s,a) P^{\infty}_{Abs}(s'|z') \tag{28}$$

$$P^{\infty}_{Abs}(z'|s,a) = \sum_{s';\ f_{s\to z}(s') = z'} P(s'|s,a) \tag{29}$$

$$P^{\infty}_{Abs}(s'|z') = \frac{1_{(f_{s\to z}(s') = z')} P^{\pi_D}(s')}{\sum_{\bar{s};\ f_{s\to z}(\bar{s}) = z'} P^{\pi_D}(\bar{s})}, \tag{30}$$

P^{π_D} is the state visitation distribution under policy π_D, and P^*_{Abs} satisfies:

$$\begin{aligned} P^*_{Abs} = &\arg\min_{\hat{P}_{Abs}} \left|\left| \sum_{s' \in S} V_{min}(s') \left| P(s'|s,a) - \hat{P}_{Abs}(s'|s,a) \right| \right|\right|_{\infty} \\ &s.t.\ \ \hat{P}_{Abs}(s'|s,a) = \sum_{\substack{z' \in Z \\ f_{s\to z}(s') = z'}} \hat{P}_{Abs}(z'|s,a) \hat{P}_{Abs}(s'|z'). \end{aligned} \tag{31}$$

Proof. See appendix. □

The bound in eq. (26) consists of four error terms, each with an important interpretation:

$$\begin{aligned} ||Q^*(s,a) - Q_{\text{Abs}}(s,a)||_{\infty} < &\text{ Structural Discrepancy} \\ &+ \text{Transition Model Estimation Variance} \\ &+ \text{Transition Model Estimation Bias} \\ &+ \text{Transition Model Class Bias} \end{aligned}$$

Structural Discrepancy: The *structural discrepancy* is defined in eq. (25) and measures the discrepancy (or dissimilarity) between state values within each partition. By assigning states with similar expected returns to the same abstract partitions, the discrepancy is decreased. Further, by increasing the number of abstract states $|Z|$ (for example, by breaking large partitions into smaller ones), the discrepancy is also decreased. The discrepancy depends only on Z and $f_{s\to z}$, which makes it independent of any collected dataset D. Unlike the previous bound in eq. (23), the discrepancy remains constant for Z and $f_{s\to z}$. However, in practice, as more data is accumulated it is of course desirable to enlarge Z with new states. In particular, if $|Z|$ is grown large enough such that each state belongs to its own abstract state (e.g. $|Z| = |S|$) then it can be shown that this term equals zero.

Transition Model Estimation Variance: This error term is a variant of the total variation distance between $P_{\text{Abs}}(s'|s,a)$ and $P^{\infty}_{\text{Abs}}(s'|s,a)$, where each term is weighted by the minimum state-value function within each abstract state $V_{\min}(s')$. The distribution $P^{\infty}_{\text{Abs}}(s'|s,a)$ represents the most accurate $P_{\text{Abs}}(s'|s,a)$ learned under the policy π_D under the constraint that the model factorizes as $P^{\infty}_{\text{Abs}}(s'|s,a) = \sum_z P^{\infty}_{\text{Abs}}(s'|z') P^{\infty}_{\text{Abs}}(z'|s,a)$. In other words, $P^{\infty}_{\text{Abs}}(s'|s,a)$ corresponds to $P_{\text{Abs}}(s'|s,a)$ estimated on an infinite dataset D collected under the policy π_D. As such, this error term is analogous to the variance term in

the bias-variance decomposition for supervised learning problems. Furthermore, suppose that $P^{\infty}_{\text{Abs}} = P^{*}_{\text{Abs}} = P$. In this case, the last two error terms in the bound are exactly zero, and this error term is smaller than the general bound in eq. (23) since applying $V_{\min}(s') \leq r_{\max}/(1-\gamma)$ yields:

$$\begin{aligned}
&\frac{\gamma}{1-\gamma}\left|\left|\sum_{s' \in S} V_{\min}(s')\left|P_{\text{Abs}}(s'|s,a) - P^{\infty}_{\text{Abs}}(s'|s,a)\right|\right|\right|_{\infty} \\
&\leq \frac{\gamma}{1-\gamma}\left|\left|\sum_{s' \in S} \frac{r_{\max}}{1-\gamma}\left|P_{\text{Abs}}(s'|s,a) - P^{\infty}_{\text{Abs}}(s'|s,a)\right|\right|\right|_{\infty} \\
&= \frac{\gamma r_{\max}}{(1-\gamma)^2}\left|\left|\sum_{s' \in S}\left|P_{\text{Abs}}(s'|s,a) - P(s'|s,a)\right|\right|\right|_{\infty}
\end{aligned}$$

For problems with large state spaces or with extreme state values, we might expect $V_{\min}(s') << r_{\max}/(1-\gamma)$ for the majority of states $s' \in S$. In this case, this bound would be far smaller than the general bound given in eq. (23). Finally, we may observe the sampling complexity of this error term. Under the simple counting distribution introduced earlier, P_{Abs} has $|S||A||Z| + |Z||S|$ parameters. This suggests only $O(|S||Z||A|)$ samples are required to reach a certain accuracy. In contrast, for the general P_{Approx} with a counting distribution, the sampling complexity grows with $O(|S|^2|A|)$. For $|Z| << |S|$, we would expect this term to decrease on the order of $O(|S|)$ times faster than the term in eq. (23).

Transition Model Estimation Bias: This error term measures the weighted total variation distance between $P^{\infty}_{\text{Abs}}(s'|s,a)$ and $P^{*}_{\text{Abs}}(s'|s,a)$, where each term is weighted by $V_{\min}(s')$. In other words, it measures the distance between the most accurate approximate transition distribution $P_{\text{Abs}}(s'|s,a)$, obtainable from an infinite dataset D collected under policy π_D, and the optimal factorized transition distribution $P^{*}_{\text{Abs}}(s'|s,a)$ (i.e. the transition distribution with the minimum sum of weighted absolute differences to the true transition distribution). As such, this error term represents the systematic bias induced by the behaviour policy π_D.

Transition Model Class Bias: This error term measures the weighted total variation distance between $P^{*}_{\text{Abs}}(s'|s,a)$ and $P(s'|s,a)$, where each term is weighted by $V_{\min}(s')$. It represents the systematic bias induced by the restricted class of probability distributions, which factorize according to latent abstract states with the mapping $f_{s \to z}$. As such, this error term is analogous to the bias term in the bias-variance decomposition for supervised learning problems. As more data is accumulated it is possible to enlarge Z with new states. In particular, if $|Z|$ is grown large enough, such that each state belongs to its own abstract state, and if $P_{\text{Abs}}(s'|s,a)$ is a tabular function, then this error term will become zero.

The bound in eq. (26) offers more than a theoretical analysis. In the extreme case where $|S| >> |D|$, the bound inspires hope that we may yet learn an effective policy if only we can learn an abstraction with small *structural discrepancy*.

5. Experiments

We carry out experiments on two natural language processing tasks in order to evaluate the performance of the Bottleneck Simulator and to compare it to other approaches. Many

real-world natural language processing tasks involve complex, stochastic structures, which have to be modelled accurately in order to learn an effective policy. Here, large amounts of training data (e.g. training signals for on-policy learning, or observed trajectories of human agents executing similar tasks) are often not available. This makes these tasks particularly suitable for demonstrating the advantages of the Bottleneck Simulator related to data-efficiency, including improved performance based on few samples.

5.1 Text Adventure Game

The first task is the text adventure game *Home World* introduced by Narasimhan, Kulkarni, and Barzilay (2015). The environment consists of four connected rooms: *kitchen*, *bedroom*, *garden* and *living room*. The game's objective involves executing a given task in a specific room, such as *eating* an *apple* in the *kitchen* when the task objective is *"You are hungry"*. The agent receives a reward of 1.0 once the task is completed. Further, we adopt the more complex setting where the objectives are redundant and confusing, such as *"You are not hungry but now you are sleepy."*. The vocabulary size is 84. The environment has 192 unique states and 22 actions.

Setup: We use the same experimental setup and hyper-parameters as Narasimhan et al. (2015) for our baseline. We train an state-action-value function baseline policy parametrized as a feed-forward neural network with Q-learning. The baseline policy is trained until the average percentage of games completed reaches at least either 15%, 20% or 30%. Then, we estimate the Bottleneck Simulator environment model with the episodes collected thus far (~1500 transitions). On the collected episodes, we learn the mapping from states to abstract states, $f_{s \to z}$, by applying k-means clustering using Euclidean distance to the word embeddings computed on the objective text and current observation text. We use Glove word embeddings (Pennington, Socher, & Manning, 2014). We train the transition model on the collected episodes. The transition model is a two-layer MLP classifier predicting a probability for each cluster-id of the next state given a state and action. Finally, we train a two-layer MLP predicting the reward given a state and action. This MLP defines the reward function in the Bottleneck Simulator environment model.

Policy: We initialize the Bottleneck Simulator policy from the baseline policy and continue training it by rolling out simulations in the Bottleneck Simulator environment model. For every 150 rollouts Bottleneck Simulator environment model, we evaluate the policy in the real game by letting the agent play out 20 episodes and measure the percentage of completed games. We stop training when the percentage of completed games stops improving.

Benchmark Policies: We compare the Bottleneck Simulator policy to two benchmark policies. The first is the baseline policy trained with Q-learning. The second is a policy trained with a state abstraction method, which we call State Abstraction. The observed states $s \in S$ are mapped to abstract states $z \in Z$, where Z are the same set of abstract states utilized by the Bottleneck Simulator environment model. As with the Bottleneck Simulator environment model, the function $f_{s \to z}$ is used to map from states to abstract states. The action space $a \in A$ was not modified. As with the Bottleneck Simulator policy, we evaluate the State Abstraction policy every 150 episodes by letting the agent play out another 20

episodes and measure the percentage of completed games. For the final evaluation, we select the State Abstraction policy which obtained the highest percentage of completed games.

Evaluation: The results are given in Table 1, averaged over 10 runs for different cluster sizes. We evaluate the policies based on the percentage of games completed. It is important to note that since our goal is to evaluate sample efficiency, our policies have been trained on far fewer episodes compared to Narasimhan et al. (2015).[3] Further, we have retained the baseline policy's hyper-parameters for the Bottleneck Simulator policies while reporting the results. For the first experiment, where the baseline policy reaches 15% completed games, we observe peak performance at 36.8% for the Bottleneck Simulator policy with a cluster size of 24, which is significantly higher than the State Abstraction policy at 26.5% and the baseline policy at only 15%. For the second experiment, where the baseline policy reaches 20.7% completed games, we observe peak performance at 42.6% for the Bottleneck Simulator policy with a cluster size of 32, which is significantly higher than the State Abstraction policy at 28.4% and the baseline policy at only 20.7%. For the second experiment, where the baseline policy reaches 30.1% completed games, we observe peak performance at 63.3% for the Bottleneck Simulator policy with a cluster size of 32, which is significantly higher than the State Abstraction policy at 57.5% and the baseline policy at only 30.1%. This shows empirically that the Bottleneck Simulator policy is the most sample efficient algorithm, since all policies have been trained on the same number of examples. Furthermore, we observe that the performance of both the Bottleneck Simulator policy and the State Abstraction policy improve as the number of completed games of the Q-learning baseline policy increases. This is to be as expected, since a higher number of completed games of the Q-learning baseline policy means a larger dataset available for training Bottleneck Simulator policy and the baseline policy. Finally, it should be noted that the State Abstraction and Bottleneck Simulator policies are complementary and may even be combined (e.g. by training a State Abstraction policy from samples generated by the Bottleneck Simulator environment model).

5.2 Dialogue

The second task is a real-world problem, where the agent must select appropriate responses in social, chit-chat conversations. The task is the 2017 Amazon Alexa Prize Competition (Ram, Prasad, Khatri, Venkatesh, et al., 2017), where a spoken dialogue system must converse coherently and engagingly with humans on popular topics (e.g. entertainment, fashion, politics, sports).[4][5]

Setup: We experiment with a dialogue system consisting of an ensemble of 22 *response models*. The response models take as input a dialogue and output responses in natural language text. In addition, the response models may also output one or several scalar values, indicating confidence levels. The response models have each their own internal procedure for generating responses: some response models are based on information retrieval models, others on generative language models, and yet others on template-based procedures. Taken together, these response models output a diverse set of responses. The dialogue system is

3. Indeed, a tabular state-action-value function could straightforwardly be trained with Q-learning to solve this task perfectly if given enough training examples.
4. See also https://developer.amazon.com/alexaprize/2017-alexa-prize.
5. Two demo videos of the system are available online: https://youtu.be/TCVbYpu9Llo and https://youtu.be/LG482LzW77Y.

Table 1: Average percentage of completed games for *Home World* (± 95% confidence intervals) across three different experiments. The Q-learning baseline policy was trained once until reaching an average game completion rate of 15%, 20.7% and 30.1% in the first, second and third experiment respectively.

		Number of Clusters (i.e. $\|Z\|$)			
	Policy	**4**	**16**	**24**	**32**
Exp 1	Q-learning (Baseline)	15.0	15.0	15.0	15.0
	State Abstraction	27.8±4.4	24.7±8.4	26.5±7.5	21.3±10.7
	Bottleneck Simulator	17.0±2.3	24.8±2.4	**36.8±2.1**	29.8±4.7
Exp 2	Q-learning (Baseline)	20.7	20.7	20.7	20.7
	State Abstraction	16.0±3.1	24.7±1.9	28.0±4.2	28.4±1.2
	Bottleneck Simulator	24.4±1.1	37.9±2.1	41.6±5.4	**42.6±3.2**
Exp 3	Q-learning (Baseline)	30.1	30.1	30.1	30.1
	State Abstraction	35.3±0.9	50.5±8.4	56.1±7.1	57.5±1.3
	Bottleneck Simulator	32.1±0.8	53.7±5.1	52.0±5.4	**63.3±3.2**

described further in Serban, Sankar, Zhang, Lin, Subramanian, Kim, Chandar, Ke, et al. (2017a) (see also Serban, Sankar, Germain, Zhang, Lin, Subramanian, Kim, Pieper, Chandar, Ke, et al. (2017b)).

The agent's task is to select an appropriate response from the set of responses, in order to maximize the satisfaction of the human user. At the end of each dialogue, the user gives a score between 1 (low satisfaction) and 5 (high satisfaction).

Prior to this experiment, a few thousand dialogues were recorded between users and two other agents acting with ϵ-greedy exploration. These dialogues are used for training the Bottleneck Simulator and the benchmark policies. In addition, about 200,000 labels were annotated at the dialogue-turn-level using crowd-sourcing: for each recorded dialogue, an annotator was shown a dialogue and several system responses (the actual response selected by the agent and several alternative responses) and asked to score each between 1 (very poor) and 5 (excellent).

Policy: The Bottleneck Simulator policy is trained using discounted Q-learning on rollout simulations from the Bottleneck Simulator environment model. The policy is parametrized as an state-action-value function $Q(s, a)$, taking as input the dialogue history s and a candidate response a. Based on the dialogue history s and candidate response a, 1458 features are computed, including word embeddings, dialogue acts, part-of-speech tags, unigram and bigram word overlap, and model-specific features. These features are given as input to a five-layered feed-forward neural network, which then outputs the estimated state-action value. Further details on the model architecture are given in the appendix.

Abstraction Space: As defined earlier, let Z be the set of abstract states used by the Bottleneck Simulator environment model. We then define Z as the Cartesian product:

$$Z = Z_{\text{Dialogue act}} \times Z_{\text{User sentiment}} \times Z_{\text{Generic user utterance}},$$

where $Z_{\text{Dialogue act}}$, $Z_{\text{User sentiment}}$ and $Z_{\text{Generic user utterance}}$ are three discrete sets. The first set consists of 10 dialogue acts, representing high-level user intentions (Stolcke, Ries, Coccaro, Shriberg, Bates, Jurafsky, Taylor, Martin, Van Ess-Dykema, & Meteer, 2000): $Z_{\text{Dialogue act}} = \{\text{Accept}, \text{Reject}, \text{Request}, \text{Politics}, \text{Generic Question},$
$\text{Personal Question}, \text{Statement}, \text{Greeting}, \text{Goodbye}, \text{Other}\}$. These dialogue acts represent the high-level intention of the user's utterance. The second set consists of sentiments types: $Z_{\text{User sentiment}} = \{\text{Negative}, \text{Neutral}, \text{Positive}\}$. The third set contains the binary variable: $Z_{\text{Generic user utterance}} = \{\text{True}, \text{False}\}$. This variable is *True* only when the user utterance is generic and topic-independent (i.e. when the user utterance only contains stop-words). We develop a deterministic classifier $f_{s \to z}$ mapping dialogue histories to corresponding classes in $Z_{\text{Dialogue act}}$, $Z_{\text{User sentiment}}$ and $Z_{\text{Generic user utterance}}$. Although we only consider dialogue acts, sentiment and generic utterances, it is trivial to expand the abstract state with other types of information.

Transition Model: The Bottleneck Simulator environment model uses a transition distribution parametrized by three independent two-layer MLP models. All three MLP models take as input the same features as the Bottleneck Simulator policy, as well as features related to the dialogue act, sentiment and generic property of the last user utterance. The first MLP predicts the next dialogue act ($Z_{\text{Dialogue act}}$), the second MLP predicts the next sentiment type ($Z_{\text{User sentiment}}$) and the third MLP predicts whether the next user utterance is generic ($Z_{\text{Generic user utterance}}$). The training dataset consists of $\sim 500,000$ recorded dialogue transitions, of which 70% of the dialogues are used as training set and 30% of the dialogues are used as validation set. The MLPs are trained with cross-entropy using mini-batch stochastic gradient descent. During rollout simulations, given a dialogue history s_t and an action a_t selected by the policy, the next abstract state $z_{t+1} \in Z$ is sampled according to the predicted probability distributions of the three MLP models. Then, a corresponding next dialogue history s_{t+1} is sampled at uniformly random from the set of recorded dialogues, under the constraint that the dialogue history matches the abstract state (i.e. $f_{s \to z}(s_{t+1}) = z_{t+1}$).

Reward Model: The Bottleneck Simulator environment model uses a reward model parametrized as a feed-forward neural network with a softmax output layer. The reward model is trained to estimate the reward for each action based on the 200,000 crowd-sourced labels. When rolling out simulations with the Bottleneck Simulator, the expected reward is given to the agent at each time step. Unless otherwise stated, in the remainder of this section, this is the model we will refer to as the learned, approximate reward model.

Benchmark Policies: We evaluate the Bottleneck Simulator policy against the following seven competing methods:

Heuristic: a heuristic policy based on pre-defined rules.

Supervised: an state-action-value function policy trained with supervised learning (cross-entropy) to predict the annotated scores on the ~200,000 crowd-sourced labels.

Q-learning: an state-action-value function policy trained with discounted Q-learning on the recorded dialogues, where episode returns are given by a learned, approximate reward model.

Q-Function Approx: an state-action-value function policy trained on the. ∼500,000 recorded transitions with a least-squares regression loss, where the target values are given by a learned, approximate reward model.

REINFORCE: an off-policy REINFORCE policy trained with reward shaping on the ∼500,000 recorded transitions, where episode returns are given by the final user scores.

REINFORCE Critic: an off-policy REINFORCE policy trained with reward shaping on the ∼500,000 recorded transitions, where episode returns are given by a learned, approximate reward model.

State Abstraction: a tabular state-action-value function policy trained with discounted Q-learning on rollouts from the Bottleneck Simulator environment model, with abstract policy state space $Z = Z_{\text{Dialogue act}} \times Z_{\text{User sentiment}} \times Z_{\text{Generic user utterance}}$ containing 60 discrete abstract states and action space containing 52 abstract actions, and where episode returns are given by a learned, approximate reward model.

The two off-policy REINFORCE policies were trained with the action probabilities of the recorded dialogues (information which none of the other policies used). With the exception of the Heuristic and State Abstraction policies, all policies were parametrized as five-layered feed-forward neural networks. Furthermore, the Bottleneck Simulator, the Q-learning, the Q-Function Approx. and the two off-policy REINFORCE policies were all initialized from the Supervised policy. This is analogous to pretraining the policies to imitate a myopic human policy (i.e. imitating the immediate actions of humans in given states). For these policies, the first few hidden layers were kept fixed after initialization from the Supervised policy, due to the large number of parameters. See appendix for details.

Preliminary Evaluation: We use two methods to perform a preliminary evaluation of the learned policies. The first method evaluates each policy using the crowdsourced human scores. For each dialogue history, the policy must select one of the corresponding annotated responses. Afterwards, the policy receives the human-annotated score as reward. Finally, we compute the average human-annotated score of each policy. This evaluation serves as a useful approximation of the immediate, average reward a policy would get on the set of annotated dialogues.[6]

The second method evaluates each policy by running 500 rollout simulations in the Bottleneck Simulator environment model, and computes the average return and average reward per time step. The rollouts are carried out on the held-out validation set of dialogue transitions (i.e. only states $s \in S$, which occur in the held-out validation set are sampled during rollouts). Although the Bottleneck Simulator environment model is far from an

6. The feed-forward neural network policies were all pre-trained with cross-entropy to predict the training set of the crowdsourced labels, such that their second last layer computes the probability of each human score (see appendix for details). Therefore, the output of their last hidden layer is used to select the response in the crowdsourced evaluation. Note further that the crowdsourced evaluation is carried out on the held-out test set of crowdsourced labels, while the neural network parameters were trained on the training set of crowdsourced labels.

Table 2: Policy evaluation w.r.t. average crowdsourced scores (± 95% confidence intervals), and average return and reward per time step computed from 500 rollouts in the Bottleneck Simulator environment model (± standard deviations). Star * indicates policy is significantly better than Heuristic policy at 95% statistical significance level. Triangle ▲ indicates policy is initialized from Supervised policy feed-forward neural network and hence yield same performance w.r.t. crowdsourced human scores.

	Crowdsourced	**Simulated Rollouts**	
Policy	**Human Score**	**Return**	**Avg Reward**
Heuristic	2.25±0.04	−11.33±12.43	−0.29±0.19
Supervised	**2.63±0.05***	**−6.46±8.01**	**−0.15±0.16**
Q-learning	**2.63±0.05*▲**	−6.70±7.39	**−0.15±0.17**
Q-Function Approx.	**2.63±0.05*▲**	−24.19±23.30	−0.73±0.27
REINFORCE	**2.63±0.05*▲**	−7.30±8.90	−0.16±0.16
REINFORCE Critic	**2.63±0.05*▲**	−10.19±11.15	−0.28±0.19
State Abstraction	1.85±0.04	−13.04±13.49	−0.35±0.19
Bottleneck Sim.	**2.63±0.05*▲**	−6.54±8.02	**−0.15±0.18**

Table 3: A/B testing experiments average real-world user scores (± 95% confidence intervals). Star * indicates policy is significantly better than other policies at 95% statistical significance level. Results are based on a total of ~3000 real-world users.

Policy	**Exp 1**	**Exp 2**	**Exp 3**
Heuristic	2.86±0.22	-	-
Supervised	2.80±0.21	-	-
Q-Function Approx.	2.74±0.21	-	-
REINFORCE	2.86±0.21	**3.06±0.12**	3.03±0.18
REINFORCE Critic	2.84±0.23	-	-
Bottleneck Sim.	**3.15±0.20***	2.92±0.12	**3.06±0.17**

accurate representation of the real world, it has been trained with cross-entropy (maximum log-likelihood) on ~500,000 recorded transitions. Therefore, the rollout simulations might serve as a useful first approximation of how a policy might perform when interacting with real-world users. The exception to this interpretation is the Bottleneck Simulator and State Abstraction policies, which themselves utilized rollout simulations from the Bottleneck Simulator environment model during training. Because of this, it is possible that the these two policies might be overfitting the Bottleneck Simulator environment model and, in turn, that this evaluation might be over-estimating their performance. Therefore, we will not

Table 4: First A/B testing experiment topical specificity and coherence by policy. The columns are average number of noun phrases per system utterance (System NPs), average number of overlapping words between the user's utterance and the system's response (This Turn), and average number of overlapping words between the user's utterance and the system's response in the next turn (Next Turn). Stop words are excluded. 95% confidence intervals are also shown.

		Word Overlap	
Policy	**System NPs**	**This Turn**	**Next Turn**
Heuristic	1.05±0.05	7.33±0.21	2.99±1.37
Supervised	1.75±0.07	10.48±0.28	10.65±0.29
Q-Function Approx.	1.50±0.07	8.35±0.29	8.36±0.31
REINFORCE	1.45±0.05	9.05±0.21	9.14±0.22
REINFORCE Critic	1.04±0.06	7.42±0.25	7.42±0.26
Bottleneck Sim.	**1.98±0.08**	**11.28±0.30**	**11.52±0.32**

consider strong performance of either of these two policies here as indicating that they are superior to other policies.

The results are given in Table 2. On the crowdsourced evaluation, the Supervised policy and all policies initialized perform decently reaching an average human score of 2.63. This is to be expected, since the Supervised policy is trained only to maximize the crowdsourced human scores. However, the Heuristic policy performs significantly worse indicating that there is much improvement to be made on top of the pre-defined rules. Further, the State Abstraction policy performs worst out of all the policies, indicating that the abstract state-action space cannot effectively capture important aspects of the states and actions to learn a useful policy for this complex task. On the rollout simulation evaluation, we observe that the Supervised policy, Q-learning policy, and Bottleneck Simulator policy are tied for first place. Since the Bottleneck Simulator policy performs similarly to the other policies here, it would appear that the policy has not overfitted the Bottleneck Simulator environment model. After these policies follow the two REINFORCE policies and the Heuristic policy. Second last comes the State Abstraction policy, which again indicates that the state abstraction method is insufficient for this complex task. Finally, the Q-function Approx. appears to perform the worst, suggesting that the learned, approximate reward model it was trained with does not perform well.

This section provided a preliminary evaluation of the policies. The next section will provide a large-scale, real-world user evaluation.

Real-World Evaluation: We carry out a large-scale evaluation with real-world users through three A/B testing experiments conducted during the Amazon Alexa Prize Competition, between July 29th - August 21st, 2017. In the first experiment the Heuristic, Supervised, Q-Function Approx., REINFORCE, REINFORCE Critic and Bottleneck Simulator policies were evaluated. In the next two experiments only the Bottleneck Simulator and REINFORCE policies were evaluated. In total, ~3000 user scores were collected.

The average user scores are given in Table 3. We observe that the Bottleneck Simulator policy performed best in both the first and third experiments. This shows that the Bottleneck Simulator policy has learned an effective policy, which is in agreement with the preliminary evaluation. On the other hand, the REINFORCE policy performed best in the second experiment. This shows that the REINFORCE policy is the most fierce contender of the Bottleneck Simulator policy. In line with the preliminary evaluation, the REINFORCE Critic and Q-Function Approx. policies perform worse than the REINFORCE and Bottleneck Simulator policies. Finally, in contrast to the preliminary evaluation, the Supervised policy performs worse than all other policies evaluated, though not significantly worse than the Heuristic policy.

Next, we conduct an analysis of the policies in the first experiment w.r.t. topical specificity and topical coherence. For topical specificity, we measure the average number of noun phrases per system utterance. A topic-specific policy will score high on this metric. For topical coherence, we measure the word overlap between the user's utterance and the system's response, as well as word overlap between the user's utterance and the system's response at the next turn. The more a policy remains on topic, the higher we would expect these two metrics to be. An engaging policy should score high on both topical specificity and topical coherence.

As shown in Table 4, the Bottleneck Simulator policy performed best on all three metrics. This indicates that the Bottleneck Simulator has the most coherent and engaging dialogues out of all the evaluated policies. This is in agreement with its excellent performance w.r.t. real-world user scores and w.r.t. the preliminary evaluation. A further analysis of the selected responses indicates that the Bottleneck Simulator has learned a more *risk tolerant* strategy.

6. Related Work

This section discusses work related to the Bottleneck Simulator.

Model-Based RL: The Bottleneck Simulator is inspired by previous work on model-based RL. Model-based RL research dates back to the 90s, and includes well-known algorithms such as Dyna, R-max and E^3 (Sutton, 1990; Moore & Atkeson, 1993; Peng & Williams, 1993; Kuvayev & Sutton, 1997; Brafman & Tennenholtz, 2002; Kearns & Singh, 2002; Wiering & Schmidhuber, 1998; Wiering, Sałustowicz, & Schmidhuber, 1999). For an overview, see Kaelbling, Littman, and Moore (1996). Model-based RL with deep learning has also been investigated, in particular for robotic control tasks (Watter et al., 2015; Lenz, Knepper, & Saxena, 2015; Gu, Lillicrap, Sutskever, & Levine, 2016; Finn & Levine, 2017). Sample efficient approaches have also been proposed by taking a Bayesian approach to learning the dynamics model. For example, PILCO incorporates uncertainty by learning a distribution over models of the dynamics in conjunction with the agent's policy (Deisenroth & Rasmussen, 2011). Another approach based on Bayesian optimization was proposed by Bansal, Calandra, Xiao, Levine, and Tomiin (2017). An approach combining dynamics models of various levels of fidelity or accuracy was proposed by Cutler, Walsh, and How (2015). Other related work includes (Bansal et al., 2017; Cutler et al., 2015; Oh, Guo, Lee, Lewis, & Singh, 2015; Venkatraman, Capobianco, Pinto, Hebert, Nardi, & Bagnell, 2016; Kansky, Silver, Mely,

Eldawy, et al., 2017; Racaniere, Weber, Reichert, Buesing, et al., 2017; Francois-Lavet, Bengio, Precup, & Pineau, 2019).

A key idea employed by the Bottleneck Simulator is to group similar states together. The idea of grouping similar states together also has a long history in the RL community. Numerous algorithms exist for models based on state abstraction (state aggregation) (Bean et al., 1987; Bertsekas & Castanon, 1989; Dean et al., 1997; Dietterich, 2000; Jong & Stone, 2005; Li, Walsh, & Littman, 2006; Jiang et al., 2015). The main idea of state abstraction is to group together similar states and solve the reduced MDP. Solving the optimization problem in the reduced MDP requires far fewer iterations or samples, which improves convergence speed and sample efficiency. In particular, related theoretical analyses of the regret incurred by state abstraction methods are provided by Van Roy (2006) and Petrik and Subramanian (2014). In contrast to state abstraction, in the Bottleneck Simulator the grouping is applied exclusively within the approximate transition model while the agent's policy operates on the complete, observed state. Compared to state abstraction, the Bottleneck Simulator reduces the impact of compounding errors caused by inaccurate abstractions in the approximate transition model. By giving the policy access to the complete, observed state, it may counter inaccurate abstractions by optimizing for myopic (next-step) rewards. This enables pretraining the policy to mimic a myopically optimal policy (e.g. single human actions), as is the case in the dialogue response selection task. Furthermore, the Bottleneck Simulator allows a deep neural network policy to learn its own high-level, distributed representations of the state from scratch. Finally, the Bottleneck Simulator enables a mathematical analysis of the trade-offs incurred by the learned transition model in terms of structural discrepancy and weighted variation distances, which is not possible in the case of general, approximate transition models.

In a related vein, learning a factorized MDP (for example, by factorizing the state transition model) has also been investigated extensively in the literature (Boutilier, Dean, & Hanks, 1999; Degris, Sigaud, & Wuillemin, 2006; Strehl, Diuk, & Littman, 2007; Ross & Pineau, 2008; Bellemare, Veness, & Bowling, 2013; Wu, Feng, & Zheng, 2014; Osband & Van Roy, 2014; Hallak, Schnitzler, Mann, & Mannor, 2015). For example, Ross and Pineau (2008) develop an efficient Bayesian framework for learning factorized MDPs. As another example, Bellemare et al. (2013) propose a Bayesian framework for learning a factored environment model based on a class of recursively decomposable factorizations. An important line of work in this area are stochastic factorization models (Barreto, Beirigo, Pineau, & Precup, 2015, 2016). Similar to non-negative matrix factorization (NMF), these models approximate the environment transition model $\mathbf{P}$ with matrices $\mathbf{DK} \approx \mathbf{P}$. Similar to other methods, these models may improve sample efficiency when the intrinsic dimensionality of the transition model is low. However, in comparison to the Bottleneck Simulator and other methods, it is difficult to incorporate domain-specific knowledge since $\mathbf{D}$ and $\mathbf{K}$ are learned from scratch. In contrast to the Bottleneck Simulator and other state abstraction methods, there is no constraint for each state to belong to exactly one abstract state. Whether or not this constraint improves or deteriorates performance is task specific. However, without imposing this constraint, it seems unlikely that one can provide a mathematical analysis of policy performance in terms of structural discrepancy.

It is also instructive to compare the Bottleneck Simulator to the broader research field of hierarchical reinforcement learning (McGovern, Sutton, & Fagg, 1997; Parr & Russell,

1998b, 1998a; Hauskrecht, Meuleau, Kaelbling, Dean, & Boutilier, 1998; Sutton, Precup, & Singh, 1999; Dietterich, 2000; Barto & Mahadevan, 2003), in particular approaches where spatio-temporal abstractions are utilized in order to help break down the sequential decision making problem into smaller pieces. One such approach is the semi-Markov decision process (SMDP) with options framework (Sutton et al., 1999), where a policy is trained to operate simultaneously on primitive actions and on so-called options, a generalization of primitive actions, which includes temporally extended courses of actions. In contrast to this framework, as well as other frameworks for temporal abstractions, the Bottleneck Simulator does not construct abstractions related to the actions but utilizes instead abstraction for the state representation at a single point in time (e.g. by mapping the dialogue history at a single point in time to a corresponding abstract state). Apart from the work discussed earlier in this section, a lot of the work on hierarchical reinforcement learning seems to have focused on temporal abstractions related to the actions while keeping the state representation the same (Francois-Lavet, Henderson, Islam, Bellemare, Pineau, et al., 2018). Nevertheless, it seems likely that such temporal abstractions may be combined with the Bottleneck Simulator in order to further improve performance.

Finally, a highly relevant research direction is the model-based reinforcement learning framework of value-aware model learning (VAML) (Farahmand, Barreto, & Nikovski, 2017; Asadi, Cater, Misra, & Littman, 2018; Luo, Xu, Li, Tian, Darrell, & Ma, 2019). In this framework the environment transition model is learned by minimizing a loss function, which takes into account the impact the transition model has on the learned policy. This is in contrast to the approach we proposed above, where the Bottleneck Simulator environment model is simply trained with a cross-entropy objective function. Specifically, the loss function in the VAML framework is derived from the residual error between the Bellman operator w.r.t. the true transition model and the Bellman operator w.r.t. the learned transition model. By learning the environment transition model with such a loss function, the parameter updates of the policy at training time may become more accurate, which in turn may improve the performance of the learned policy. However, since the loss function makes further approximations and only considers the residual error arising from applying the Bellman operator once, it is not clear whether the converged, learned policy will be better compared to a policy learned using an transition model trained with a cross-entropy objective function. Nevertheless, future work should explore whether it is possible to improve the performance of the Bottleneck Simulator by incorporating the VAML framework for learning the transition model.

Dialogue Systems: Numerous researchers have applied RL for training goal-oriented dialogue systems (Singh, Kearns, Litman, & Walker, 1999; Walker, 2000; Singh, Litman, Kearns, & Walker, 2002; Williams & Young, 2007a, 2007b; Pieraccini, Suendermann, Dayanidhi, & Liscombe, 2009). One line of research has focused on learning dialogue systems through simulations, often using abstract dialogue states and actions (Eckert, Levin, & Pieraccini, 1997; Levin, Pieraccini, & Eckert, 2000; Chung, 2004; Cuayahuitl, Renals, Lemon, & Shimodaira, 2005; Georgila, Henderson, & Lemon, 2006; Schatzmann, Thomson, Weilhammer, Ye, & Young, 2007; Heeman, 2009; Traum, Marsella, Gratch, Lee, & Hartholt, 2008; Lee & Eskenazi, 2012; Khouzaimi, Laroche, & Lefevre, 2017; Lopez-Cozar, 2016; Su, Gasic, Mrksic, Rojas-Barahona, et al., 2016; Fatemi, Asri, Schulz, He, & Suleman, 2016; Asri, He, & Suleman, 2016; Kreyssig, Casanueva, Budzianowski, & Gasic, 2018). The approaches

here differ based on how the simulator is created or estimated, and whether or not the simulator is also considered an agent trying to optimize its own reward. For example, Levin et al. (2000) tackle the problem of building a flight booking dialogue system. They estimate a user simulator model by counting transition probabilities between abstract dialogue states and user actions (similar to an n-gram model), which is then used to train an RL policy. As a more recent example, Yu, Xu, Black, and Rudnicky (2016) propose to learn a dialogue manager policy through model-free off-policy RL based on simulations with a rule-based system. As another example, Kreyssig et al. (2018) propose to learn a dialogue manager policy through a user simulator, which simulates the user's response text in the dialogue, and demonstrate that this may lead to better performance compared to a user simulator invoking abstractions.

Researchers have also investigated learning generative neural network policies operating directly on raw text through user simulations (Zhao & Eskenazi, 2016; Guo, Klinger, Rosenbaum, Bigus, et al., 2016; Das, Kottur, Moura, Lee, & Batra, 2017; Lewis, Yarats, Dauphin, Parikh, & Batra, 2017; Liu & Lane, 2017). In parallel to our work, Peng, Li, Gao, Liu, and Wong (2018) have proposed a related model-based reinforcement learning approach for dialogue utilizing the Dyna algorithm. To the best of our knowledge, the Bottleneck Simulator is the first model-based RL approach with discrete, abstract states to be applied to learning a dialogue policy operating on raw text.

In a related vein, researchers have investigated learning effective dialogue managers policies by estimating a partially observable Markov decision process (POMDP) (Williams & Young, 2007b, 2007a; Gasic & Young, 2011; Png, Pineau, & Chaib-Draa, 2012; Casanueva, Budzianowski, Su, Ultes, Rojas-Barahona, Tseng, & Gasic, 2018). In both this line of work and in the Bottleneck Simulator a probability distribution over latent states is estimated. However, a major difference between the two approaches is that the POMDP approaches often require the dialogue system designer to explicitly construct the complete, latent state of the world (for example, in the form of slot-value pairs) rather than an incomplete, abstract state. The POMDP approaches often also require the dialogue system designer to put a significant amount of work into specifying, and even hand-crafting, the transition and observation functions as well as the set of possible user intentions. If the latent state of the world is missing components (e.g. slot-value pairs), if the transition and observation functions are not sufficiently accurate or if some user intentions have been omitted, then a POMDP approach may not perform well. For this reason the POMDP approaches may be very adept at learning effective policies on goal-driven dialogue tasks within clearly defined, limited domains, but may struggle to learn effective policies on non-goal-driven dialogue tasks as well as in broader domains (for example, domains where it is difficult to construct slot-value pair representations with high domain coverage). On the other hand, the Bottleneck Simulator only requires the dialogue system designer to create a small set of abstract states Z along with function $f_{s \to z}$ mapping dialogues to abstract states. There is no need for the abstract states Z to capture all salient information from a dialogue, as highlighted by the bound in eq. (26) and its dependence on the structural discrepancy. Further, as demonstrated in the first experiment, even a simple, automated approach such as clustering, may be sufficient to construct the abstract states and learn a policy superior to other reinforcement learning approaches. This also shown by the second experiment, where a Bottleneck Simulator policy

trained with only 60 possible abstract, latent states ($|Z| = 60$) manages to outperform seven competing policies.

7. Conclusion

We have proposed the Bottleneck Simulator, a model-based reinforcement learning (RL) approach combining a learned, factorized environment transition model with rollout simulations to learn an effective policy from few data examples. The learned transition model employs an abstract, discrete state (a bottleneck state), which increases sample efficiency by reducing the number of model parameters and by exploiting structural properties of the environment. We have provided a mathematical analysis of the Bottleneck Simulator in terms of fixed points of the learned policy. The analysis reveals how the policy's performance is affected by four distinct sources of errors related to the abstract space structure (structural discrepancy), to the transition model estimation variance, to the transition model estimation bias, and to the transition model class bias. We have evaluated the Bottleneck Simulator on two natural language processing tasks: a text adventure game and a real-world, complex dialogue response selection task. On both tasks, the Bottleneck Simulator has shown excellent performance beating competing approaches. In contrast to much of the previous work on abstraction in RL, our dialogue experiments are based on a complex, real-world task with a very high-dimensional state space and evaluated by real-world users.

Acknowledgments

The authors wish to thank the University of Montreal team participating in the Amazon Alexa 2017 Prize for building the ensemble-based system and infrastructure which enabled the research: Mathieu Germain, Saizheng Zhang, Zhouhan Lin, Sandeep Subramanian, Taesup Kim, Sarath Chandar, Nan Rosemary Ke, Sai Rajeshwar, Alexandre de Brebisson, Jose M. R. Sotelo, Dendi Suhubdy, Vincent Michalski and Alexandre Nguyen. The authors wish to Aaron Courville, Michael Noseworthy, Nicolas Angelard-Gontier, Ryan Lowe, Prasanna Parthasarathi and Peter Henderson for their helpful advice. Furthmore, the authors wish to thank Amazon for providing Tesla K80 GPUs through the Amazon Web Services platform. Some of the Titan X GPUs used for this research were generously donated by the NVIDIA Corporation. The authors acknowledge NSERC, Canada Research Chairs, CIFAR, IBM Research, Nuance Foundation, Microsoft Maluuba and Druide Informatique Inc. for funding. This research was enabled in part by support provided by Calcul Qubec (www.calculquebec.ca) and Compute Canada (www.computecanada.ca).

Appendix A. Dynamic Programming Preliminaries

The Bellman optimality equations can be shortened by defining the *Bellman operator* (sometimes called the *dynamic programming operator*) B (Bertsekas & Tsitsiklis, 1995, Chapter 2). For a given (not necessarily optimal) state-action-value function Q^π, the

operator is:

$$(BQ^{\pi})(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a \in A} Q^{\pi}(s',a). \tag{32}$$

In other words, the operator B updates Q towards Q^* with one dynamic programming iteration.

We need the following lemma, as derived by Jiang et al. (2015).

Lemma 1. *Let Q_1 and Q_2 be the fixed points for the Bellman optimality operators B_1, B_2, which both operate on $\mathbb{R}^{|S|\times|A|}$ and have contraction rate $\gamma \in [0,1)$:*

$$||Q_1 - Q_2||_\infty \leq \frac{||B_1 Q_2 - Q_2||_\infty}{1-\gamma}. \tag{33}$$

Proof. We prove the inequality by writing out the left-hand side, applying the triangle inequality and the Bellman residual bound (Bertsekas & Tsitsiklis, 1995, Chapter 2):

$$\begin{aligned} ||Q_1 - Q_2||_\infty &= ||Q_1 - B_1 Q_2 + B_1 Q_2 - Q_2||_\infty \\ &\leq ||Q_1 - B_1 Q_2||_\infty + ||B_1 Q_2 - Q_2||_\infty \\ &= ||B_1 Q_1 - B_1 Q_2||_\infty + ||B_1 Q_2 - Q_2||_\infty \\ &\leq \gamma ||Q_1 - Q_2||_\infty + ||B_1 Q_2 - Q_2||_\infty \end{aligned}$$

We move the first term on the right-hand side to the other side of the inequality and re-order the terms:

$$\begin{gathered} ||Q_1 - Q_2||_\infty - \gamma ||Q_1 - Q_2||_\infty \leq ||B_1 Q_2 - Q_2||_\infty \\ \Leftrightarrow \\ (1-\gamma)||Q_1 - Q_2||_\infty \leq ||B_1 Q_2 - Q_2||_\infty \\ \Leftrightarrow \\ ||Q_1 - Q_2||_\infty \leq \frac{||B_1 Q_2 - Q_2||_\infty}{1-\gamma} \end{gathered}$$

□

Appendix B. Co-Occurrence Sample Efficiency

The common, but naïve transition model estimated by eq. (8) is not very sample efficient. In order to illustrate this, assume that sample efficiency for a transition (s,a,s') is measured as the probability that $P_{\text{Approx}}(s'|s,a)$ is more than $\epsilon > 0$ away from the true value in absolute value. We bound it by Chebyshev's inequality:

$$P(|P_{\text{Approx}}(s'|s,a) - P(s'|s,a)| > \epsilon)) \leq \frac{\sigma^2}{\text{Count}(s,a,\cdot)\epsilon^2},$$

where $\sigma^2 = \max_{s,a,s'} P(s'|s,a)(1-P(s'|s,a))$ (i.e. an upper bound on the variance of a single observation from a binomial random variable). The error decreases inversely linear with

the factor $\text{Count}(s, a, \cdot)$. If we assume each sample $(s, a, s') \in D$ is drawn independently at uniform random, then the expected number of samples is: $\text{E}_D[\text{Count}(s, a, \cdot)] = |D|/(|S||A|)$. Summing over all $s' \in S$, we obtain the overall sample complexity:

$$\sum_{s'} P(|P_{\text{Approx}}(s'|s,a) - P(s'|s,a)| > \epsilon) \lesssim \frac{\sigma^2}{\epsilon^2} \frac{|S|^2|A|}{|D|},$$

which grows in the order of $O(|S|^2|A|)$. Unfortunately, this implies the simple model is highly inaccurate for many real-world applications, including natural language processing and robotics applications, where the state or action spaces are very large.

Appendix C. Theorem 1

In this section we provide the proof for Theorem 1. Let Q_{Approx} be the optimal state-action-value function w.r.t. an approximate MDP $\langle S, A, P_{\text{Approx}}, R, \gamma \rangle$, and let Q^* be the optimal state-action-value function w.r.t. the true MDP $\langle S, A, P, R, \gamma \rangle$. Let γ be their contraction rates. Then, the theorem states that:

$$\begin{aligned}
&||Q^*(s,a) - Q_{\text{Approx}}(s,a)||_\infty \\
&\quad \leq \gamma r_{\max} \left|\left| \sum_{s'} \left| P(s'|s,a) - P_{\text{Approx}}(s'|s,a) \right| \right|\right|_\infty && (34) \\
&\quad \leq \gamma r_{\max} \sqrt{2} \left|\left| \sqrt{D_{\text{KL}}(P(s'|s,a)||P_{\text{Approx}}(s'|s,a))} \right|\right|_\infty, && (35)
\end{aligned}$$

where $D_{\text{KL}}(P(s'|s,a)||P_{\text{Approx}}(s'|s,a))$ is the conditional KL-divergence between $P(s'|s,a)$ and $P_{\text{Approx}}(s'|s,a)$:

$$D_{\text{KL}}(P(s'|s,a)||P_{\text{Approx}}(s'|s,a)) \stackrel{\text{def}}{=} \sum_{s'} P(s'|s,a) \log \frac{P(s'|s,a)}{P_{\text{Approx}}(s'|s,a)} \qquad (36)$$

Proof. We start by applying Lemma 1 to the loss:

$$
\begin{aligned}
&||Q^* - Q_{\text{Approx}}||_\infty \\
&\leq \frac{1}{1-\gamma}||BQ_{\text{Approx}} - Q_{\text{Approx}}||_\infty && \text{Apply eq. (33)} \\
&= \frac{1}{1-\gamma}||R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_{\text{Approx}}(s',a') && \text{Use definition in eq. (32)} \\
&\qquad - \ Q_{\text{Approx}}(s,a)||_\infty \\
&= \frac{1}{1-\gamma}||R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_{\text{Approx}}(s',a') && \text{Fixed point:} \\
&\qquad - R(s,a) - \gamma \sum_{s'} P_{\text{Approx}}(s'|s,a) \max_{a'} Q_{\text{Approx}}(s',a')||_\infty && Q_{\text{Approx}} = B_{\text{Approx}} Q_{\text{Approx}} \\
&= \frac{\gamma}{1-\gamma}|| \sum_{s'} P(s'|s,a) \max_{a'} Q_{\text{Approx}}(s',a') && \text{Cancel } R(s,a)\text{; move } \gamma \text{ out} \\
&\qquad - \sum_{s'} P_{\text{Approx}}(s'|s,a) \max_{a'} Q_{\text{Approx}}(s',a')||_\infty \\
&= \frac{\gamma}{1-\gamma} \left|\left| \sum_{s'} \left(P(s'|s,a) - P_{\text{Approx}}(s'|s,a)\right) \max_{a'} Q_{\text{Approx}}(s',a') \right|\right|_\infty && \text{Merge sums} \\
&\leq \frac{\gamma r_{\max}}{(1-\gamma)^2} \left|\left| \sum_{s'} \left|P(s'|s,a) - P_{\text{Approx}}(s'|s,a)\right| \right|\right|_\infty && \text{Use } Q_{\text{Approx}}(s',a') \leq \frac{r_{\max}}{1-\gamma}
\end{aligned}
$$

Here, it should be noted that norms $||\cdot||_\infty$ are taken over all combinations of $s \in S, a \in A$. Next, we recognize the sum as being two times the total variation distance between $P(s'|s,a)$ and $P_{\text{Approx}}(s'|s,a)$. Thus, we apply Pinsker's inequality (Tsybakov, 2009, p. 132) to obtain:

$$
\begin{aligned}
||Q^* - Q_{\text{Approx}}||_\infty &\leq \frac{\gamma r_{\max}}{(1-\gamma)^2} \left|\left| \sum_{s'} \left|P(s'|s,a) - P_{\text{Approx}}(s'|s,a)\right| \right|\right|_\infty \\
&\leq \frac{\gamma r_{\max}}{(1-\gamma)^2} \left|\left| 2\sqrt{\frac{1}{2} D_{\text{KL}}(P(s'|s,a)||P_{\text{Approx}}(s'|s,a))} \right|\right|_\infty \\
&= \frac{\gamma r_{\max}\sqrt{2}}{(1-\gamma)^2} \left|\left| \sqrt{D_{\text{KL}}(P(s'|s,a)||P_{\text{Approx}}(s'|s,a))} \right|\right|_\infty
\end{aligned}
$$

□

Appendix D. Theorem 2

In this section we provide the proof for Theorem 2. Let Q_{Abs} be the optimal state-action-value function w.r.t. the Bottleneck Simulator $\langle Z, S, A, P_{\text{Abs}}, R, \gamma \rangle$, and let Q^* be the optimal state-action-value function w.r.t. the true MDP $\langle S, A, P, R, \gamma \rangle$. Let γ be their contraction rates. Finally, define:

$$
\epsilon = \max_{s_i, s_j \in S;\ f_{s \to z}(s_i) = f_{s \to z}(s_j)} |V^*(s_i) - V^*(s_j)| \tag{37}
$$

Then the theorem states that:

$$
\begin{aligned}
&||Q^*(s,a) - Q_{\text{Abs}}(s,a)||_\infty \\
&< \frac{2\gamma\epsilon}{(1-\gamma)^2} \qquad (38)\\
&+ \frac{\gamma}{(1-\gamma)^2}\left|\left|\sum_{s'\in S} V_{\min}(s')\left|P_{\text{Abs}}(s'|s,a) - P^{\infty}_{\text{Abs}}(s'|s,a)\right|\right|\right|_\infty \\
&+ \frac{\gamma}{(1-\gamma)^2}\left|\left|\sum_{s'\in S} V_{\min}(s')\left|P^{\infty}_{\text{Abs}}(s'|s,a) - P^{*}_{\text{Abs}}(s'|s,a)\right|\right|\right|_\infty \qquad (39)\\
&+ \frac{\gamma}{(1-\gamma)^2}\left|\left|\sum_{s'\in S} V_{\min}(s')\left|P^{*}_{\text{Abs}}(s'|s,a) - P(s'|s,a)\right|\right|\right|_\infty
\end{aligned}
$$

where $V_{\min}$ is defined as

$$V_{\min}(s) = \min_{\substack{s'\in S,\\ f_{s\to z}(s')=f_{s\to z}(s)}} V^*(s') \qquad (40)$$

and P^{∞}_{Abs} is defined as

$$P^{\infty}_{\text{Abs}}(s'|s,a) = \sum_{z\in Z} P^{\infty}_{\text{Abs}}(z'|s,a) P^{\infty}_{\text{Abs}}(s'|z') \qquad (41)$$

$$P^{\infty}_{\text{Abs}}(z'|s,a) = \sum_{s';\ f_{s\to z}(s')=z'} P(s'|s,a) \qquad (42)$$

$$P^{\infty}_{\text{Abs}}(s'|z') = \frac{1_{(f_{s\to z}(s')=z')} P^{\pi_D}(s')}{\sum_{\bar{s};\ f_{s\to z}(\bar{s})=z'} P^{\pi_D}(\bar{s})}, \qquad (43)$$

and P^{*}_{Abs} satisfies:

$$
\begin{aligned}
P^{*}_{\text{Abs}} = &\arg\min_{\hat{P}_{\text{Abs}}} \left|\left|\sum_{s'\in S} V_{\min}(s')\left|P(s'|s,a) - \hat{P}_{\text{Abs}}(s'|s,a)\right|\right|\right|_\infty \\
&\text{s.t.}\ \ \hat{P}_{\text{Abs}}(s'|s,a) = \sum_{\substack{z'\in Z\\ f_{s\to z}(s')=z'}} \hat{P}_{\text{Abs}}(z'|s,a)\hat{P}_{\text{Abs}}(s'|z'). \qquad (44)
\end{aligned}
$$

Proof. We start by applying Lemma 1 to the loss:

$$
\begin{aligned}
&||Q^* - Q_{\text{Abs}}||_\infty \\
&\leq \frac{1}{1-\gamma} ||Q^* - B_{\text{Abs}} Q^*||_\infty && \text{Use Lemma 1} \\
&= \frac{1}{1-\gamma} \left|\left| Q^*(s,a) - R(s,a) - \gamma \sum_{s'} P_{\text{Abs}}(s'|s,a) \max_{a'} Q^*(s',a') \right|\right|_\infty && \text{Use eq. (32)} \\
&= \frac{1}{1-\gamma} \Bigg|\Bigg| R(s,a) - \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') && \text{Use } BQ^* = Q^* \\
&\qquad - R(s,a) - \gamma \sum_{s'} P_{\text{Abs}}(s'|s,a) \max_{a'} Q^*(s',a') \Bigg|\Bigg|_\infty \\
&= \frac{\gamma}{1-\gamma} \Bigg|\Bigg| \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') && \text{Reorder terms} \\
&\qquad - P_{\text{Abs}}(s'|s,a) \max_{a'} Q^*(s',a') \Bigg|\Bigg|_\infty && \text{\& move } \gamma \text{ out} \\
&= \frac{\gamma}{1-\gamma} \left|\left| \sum_{s'} P(s'|s,a) V^*(s') - P_{\text{Abs}}(s'|s,a) V^*(s') \right|\right|_\infty && \text{Insert } V^*(s') \\
&\leq \frac{\gamma}{1-\gamma} \left|\left| \sum_{s'} \left| P(s'|s,a) V^*(s') - P_{\text{Abs}}(s'|s,a) V^*(s') \right| \right|\right|_\infty && |\textstyle\sum_i a_i| \leq \sum_i |a_i|
\end{aligned}
$$

By observing that s' belongs to exactly one z' we can rewrite this to:

$$
\begin{aligned}
&= \frac{\gamma}{1-\gamma} \left|\left| \sum_{z'} \sum_{s'; f_{s\to z}(s')=z'} \left| P(s'|s,a) V^*(s') - P_{\text{Abs}}(s'|s,a) V^*(s') \right| \right|\right|_\infty \\
&= \frac{\gamma}{1-\gamma} \Bigg|\Bigg| \sum_{z'} \sum_{s'; f_{s\to z}(s')=z'} \Big| P(s'|s,a) V^*(s') - P_{\text{Abs}}(s'|s,a) V^*(s') && \text{Add \& subtract:} \\
&\qquad + P(s'|s,a) V_{\min}(s') - P(s'|s,a) V_{\min}(s') && P(s'|s,a)V_{\min}(s') \\
&\qquad + P_{\text{Abs}}(s'|s,a) V_{\min}(s') - P_{\text{Abs}}(s'|s,a) V_{\min}(s') \Big| \Bigg|\Bigg|_\infty && P_{\text{Abs}}(s'|s,a)V_{\min}(s')
\end{aligned}
$$

$$= \frac{\gamma}{1-\gamma} \Bigg|\Bigg| \sum_{z'} \sum_{s';f_{s\to z}(s')=z'} \Big|P(s'|s,a)V^*(s') - P(s'|s,a)V_{\min}(s')\Big| \qquad \text{Use}$$
$$+ \Big|P_{\text{Abs}}(s'|s,a)V^*(s') - P_{\text{Abs}}(s'|s,a)V_{\min}(s')\Big| \qquad \text{triangle inequality}$$
$$+ \Big|P(s'|s,a)V_{\min}(s') - P_{\text{Abs}}(s'|s,a)V_{\min}(s')\Big| \Bigg|\Bigg|_\infty$$
$$= \frac{\gamma}{1-\gamma} \Bigg|\Bigg| \sum_{z'} \sum_{s';f_{s\to z}(s')=z'} P(s'|s,a)\Big|V^*(s') - V_{\min}(s')\Big| \qquad \text{Combine terms}$$
$$+ P_{\text{Abs}}(s'|s,a)\Big|V^*(s') - V_{\min}(s')\Big|$$
$$+ V_{\min}(s')\Big|P(s'|s,a) - P_{\text{Abs}}(s'|s,a)\Big| \Bigg|\Bigg|_\infty$$

We now apply the assumption on similarity between states $s, s' \in S$ belonging to same abstract state $z' \in Z$ (eq. (37)):

$$\frac{\gamma}{1-\gamma} \Bigg|\Bigg| \sum_{z'} \sum_{s';f_{s\to z}(s')=z'} P(s'|s,a)\Big|V^*(s') - V_{\min}(s')\Big|$$
$$+ P_{\text{Abs}}(s'|s,a)\Big|V^*(s') - V_{\min}(s')\Big|$$
$$+ V_{\min}(s')\Big|P(s'|s,a) - P_{\text{Abs}}(s'|s,a)\Big| \Bigg|\Bigg|_\infty$$
$$< \frac{\gamma}{1-\gamma} \Bigg|\Bigg| \sum_{z'} \sum_{s';f_{s\to z}(s')=z'} P(s'|s,a)\epsilon \qquad \text{Use eq. (25) twice}$$
$$+ P_{\text{Abs}}(s'|s,a)\epsilon$$
$$+ V_{\min}(s')\Big|P(s'|s,a) - P_{\text{Abs}}(s'|s,a)\Big| \Bigg|\Bigg|_\infty$$
$$= \frac{\gamma}{1-\gamma} \Bigg|\Bigg| 2\epsilon + \sum_{z'} \sum_{s';f_{s\to z}(s')=z'} V_{\min}(s')\Big|P(s'|s,a) - P_{\text{Abs}}(s'|s,a)\Big| \Bigg|\Bigg|_\infty \qquad 1=\sum_{s'} P(s'|s,a)$$
$$=\sum_{s'} P_{\text{Abs}}(s'|s,a)$$

$$\leq \frac{2\gamma\epsilon}{1-\gamma} + \frac{\gamma}{1-\gamma}\left|\left|\sum_{z'}\sum_{s';f_{s\to z}(s')=z'} V_{\min}(s')\Big|P(s'|s,a) - P_{\text{Abs}}(s'|s,a)\Big|\right|\right|_\infty \quad \text{Triangle inequality}$$

$$= \frac{2\gamma\epsilon}{1-\gamma} + \frac{\gamma}{1-\gamma}\left|\left|\sum_{s'} V_{\min}(s')\Big|P(s'|s,a) - P_{\text{Abs}}(s'|s,a)\Big|\right|\right|_\infty \quad \text{Contract sums}$$

$$\leq \frac{2\gamma\epsilon}{1-\gamma} + \frac{\gamma}{1-\gamma}\left|\left|\sum_{s'} V_{\min}(s')\Big|P_{\text{Abs}}(s'|s,a) - P^{\infty}_{\text{Abs}}(s'|s,a)\Big|\right|\right|_\infty \quad \text{Triangle inequality}$$

$$+ \frac{\gamma}{1-\gamma}\left|\left|\sum_{s'} V_{\min}(s')\Big|P(s'|s,a) - P^{\infty}_{\text{Abs}}(s'|s,a)\Big|\right|\right|_\infty \quad \text{by inserting } P^{\infty}_{\text{Abs}}$$

Finally, we apply the triangle inequality one last time by inserting P^{*}_{Abs} in order to obtain the final result. □

Appendix E. Dialogue Experiment Benchmarks

As discussed in the Experiments section, we compare the Bottleneck Simulator to several competing approaches.

Heuristic Policy

The first approach is a heuristic policy, which selects its response from two response models in the system. The first response model is the chatbot *Alice* (Wallace, 2009; Shawar & Atwell, 2007), which generates responses by using thousands of template rules. The second response model is the question-answering system *Evi*, which is capable of handling a large variety of factual questions.[7]

A few pre-defined rules are used to decide if the user's utterance should be classified as a question or not. If it is classified as a question, the policy will select the response generated by the question-answering system *Evi*. Otherwise, the policy will select the response generated by the chatbot *Alice*.

Evi is an industry-strength question-answering system using dozens of factual databases and proprietary algorithms built over the course of an entire decade. Further, *Alice* is capable of handling many different conversations effectively using its internal database containing thousands of template rules. Therefore, this policy should be considered a strong baseline.

Supervised Policy: Learning with Crowdsourced Labels

The second approach to learning a policy is based on estimating the state-action-value function using supervised learning on crowdsourced labels. This approach also serves as initialization for the approaches discussed later.

Crowdsourcing: We use Amazon Mechanical Turk (AMT) to collect training data. We follow a setup similar to Liu, Lowe, Serban, Noseworthy, Charlin, and Pineau (2016). We show human evaluators a dialogue along with 4 candidate responses, and ask them to

7. www.evi.com.

score how appropriate each candidate response is on a 1-5 Likert-type scale. The score 1 indicates that the response is inappropriate or does not make sense, 3 indicates that the response is acceptable, and 5 indicates that the response is excellent and highly appropriate. The dialogues are extracted from interactions between Alexa users and preliminary versions of our system. For each dialogue, the corresponding candidate responses are created by generating candidate responses from the 22 response models in the system. We preprocess the dialogues and candidate responses by masking profanities and swear words. Furthermore, we anonymize the dialogues and candidate responses by replacing first names with randomly selected gender-neutral names. Finally, dialogues are truncated to the last 4 utterances and last 500 words, in order to reduce the cognitive load of the task.

After the crowdsourcing, we manually inspected the annotations and observed that annotators tended to frequently overrate topic-independent, generic responses. We corrected for this by decreasing the label scores of generic responses.

In total, we collected 199,678 labels. These are split into training (train), development (dev) and testing (test) sets consisting of respectively 137,549, 23,298 and 38,831 labels each.

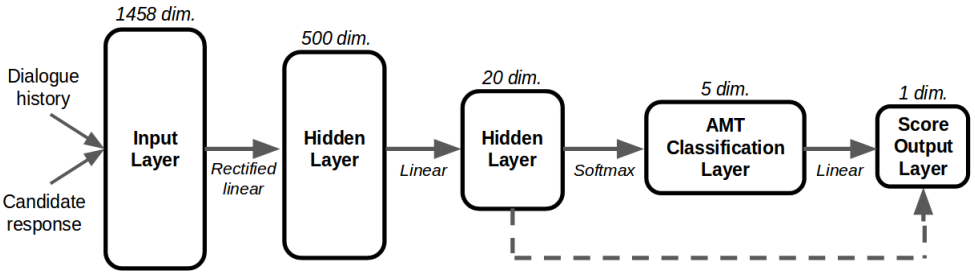

Figure 2: The system policy is parametrized as a five-layer neural network, which takes as input a dialogue history and candidate response and outputs either the estimated expected return or score. The model consists of an input layer with 1458 features, a hidden layer with 500 hidden units, a hidden layer with 20 hidden units, a softmax layer with 5 output probabilities (corresponding to the five AMT labels discussed in Section 7), and a scalar-valued output layer. The dashed arrow indicates a skip connection.

Training: The policy is parametrized as a neural network taking as input 1458 features computed based on the dialogue history and candidate response. See Figure 2. The neural network parametrizes the state-action-value function (i.e. the estimate of the expected return given a state particular state and action). We optimize the neural network parameters w.r.t. log-likelihood (cross-entropy) to predict the 4th layer, which represents the AMT label classes. Formally, we optimize the model parameters θ as:

$$\hat{\theta} = \arg\max_{\theta} \sum_{x,y} \log P_\theta(y|x), \tag{45}$$

where x are the input features, y is the corresponding AMT label class (a one-hot vector) and $P_\theta(y|x)$ is the model's predicted probability of y given x, computed in the second

last layer of the model. We use the first-order gradient-descent optimizer Adam (Kingma & Ba, 2015) . We experiment with a variety of hyper-parameters, and select the best hyper-parameter combination based on the log-likelihood of the dev set. For the first hidden layer, we experiment with layer sizes in the set: $\{500, 200, 50\}$. For the second hidden layer, we experiment with layer sizes in the set: $\{50, 20, 5\}$. We use L2 regularization on all model parameters, except for bias parameters. We experiment with L2 regularization coefficients in the set: $\{10.0, 1.0, 10^{-1}, \ldots, 10^{-9}\}$. Unfortunately, we do not have labels to train the last layer. Therefore, we fix the parameters of the last layer to the vector $[1.0, 2.0, 3.0, 4.0, 5.0]$. In other words, we assign a score of 1.0 for the label *very poor*, a score of 2.0 for the label *poor*, a score of 3.0 for the label *acceptable*, a score of 4.0 for the label *good* and a score of 5.0 for the label *excellent*. At every turn in the dialogue, the policy picks the candidate response with the highest estimated score. As this policy was trained on crowdsourced data using supervised learning, we call it *Supervised*.

Q-learning Policy

In the second approach, we fixed the last output layer parameters to $[1.0, 2.0, 3.0, 4.0, 5.0]$. In other words, we assigned a score of 1.0 for *very poor* responses, 2.0 for *poor* responses, 3.0 for *acceptable* responses, and so on. It's not clear whether this score is correlated with scores given by real-world Alexa users, which is what we ultimately want to optimize the system for. This section describes a (deep) Q-learning approach, which directly optimizes the policy towards improving the Alexa user scores.

Q-learning: Let Q be the approximate, state-action-value function parametrized by parameters θ. Let $\{s_t^d, a_t^d, R^d\}_{d,t}$ be a set of observed (recorded) examples, where s_t^d is the dialogue history for dialogue d at time step (turn) t, a_t^d is the agent's action for dialogue d at time step (turn) t and R^d is the return for dialogue d. Let D be the number of observed dialogues and let T^d be the number of turns in dialogue d. Q-learning then optimizes the state-action-value function parameters by minimizing the squared error:

$$\sum_{d=1}^{D} \sum_{t=1}^{T^d} ||Q_\theta(s_t^d, a_t^d) - r_t^d + \gamma \max_a Q_\theta(s_{t+1}^d, a)||^2 \tag{46}$$

Training: We initialize the policy with the parameters of the *Supervised* policy, and then train the parameters w.r.t. eq. (46) with stochastic gradient descent using Adam. We use a set of a few thousand dialogues recorded between users and a preliminary version of the system. The same set of recorded dialogues were used by the Bottleneck Simulator policy. About 80% of these examples are used for training and about 20% are used for development. To reduce the risk of overfitting, we only train the parameters of the second last layer. We select the hyper-parameters with the highest expected return on the development set. We call this policy *Q-learning*.

Q-Function Policy

This section describes an alternative approach to learn the state-action-value function, based on training an approximate, reward model capable of predicting the Alexa user score.

Approximate State-Action-Value Function: For time (turn) t, let s_t be a dialogue history and let a_t be the corresponding response given by the system. We aim to learn a

regression model, g_ϕ, which predicts the final return (user score) at the current turn:

$$g_\phi(s_t, a_t) \in [1, 5], \tag{47}$$

where ϕ are the model parameters. We call this an *approximate state-action-value function* or *reward model*, since it directly models the user score, which we aim to maximize. Let $\{s_t^d, a_t^d, R^d\}_{d,t}$ be a set of observed (recorded) examples, where t denotes the time step (turn) and d denotes the dialogue. Let $R^d \in [1, 5]$ denote the observed real-valued return for dialogue d. The majority of users give whole number (integer) scores, but some users give decimal scores (e.g. 3.5). Therefore, we treat R^d as a real-valued number in the range 1-5. We learn the model parameters ϕ by minimizing the squared error between the model's prediction and the observed return:

$$\hat{\phi} = \arg\max_\phi \sum_d \sum_t (g_\phi(s_t^d, a_t^d) - R^d)^2 \tag{48}$$

As before, we optimize the model parameters using mini-batch stochastic gradient descent with the optimizer Adam. We use L2 regularization with coefficients in the set $\{10.0, 1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.0\}$. We select the coefficient with the smallest squared error on a hold-out dataset.

As input to the reward model we compute 23 higher-level features based on the dialogue history and a candidate response. In total, our dataset for training the reward model has 4340 dialogues. We split this into a training set with 3255 examples and a test set with 1085 examples.

To increase sample efficiency, we learn an ensemble model through a variant of the bagging technique (Breiman, 1996). We create 5 new training sets, which are shuffled versions of the original training set. Each shuffled dataset is split into a sub-training set and sub-hold-out set. The sub-hold-out sets are created such that the examples in one set do not overlap with other sub-hold-out sets. A reward model is trained on each sub-training set, with its hyper-parameters selected on the sub-hold-out set. This increases sample efficiency by allowing us to re-use the sub-hold-out sets for training, which would otherwise not have been used. The final reward model is an ensemble, where the output is an average of the underlying linear regression models.

Training: As with the supervised learning approach, the policy here is a neural network which parametrizes an state-action-value function. To prevent overfitting, we do not train the neural network from scratch with the reward model as target. Instead, we initialize the model with the parameters of the *Supervised* neural network, and then fine-tune it with the reward model outputs to minimize the squared error:

$$\hat{\theta} = \arg\max_\theta \sum_d \sum_t (f_\theta(s_t^d, a_t^d) - g_\phi(s_t^d, a_t^d))^2, \tag{49}$$

As before, we optimize the model parameters using mini-batch stochastic gradient descent with Adam. As training this model does not depend on AMT labels, training is carried out on recorded dialogues. We train on several thousand recorded dialogue examples, where about 80% are used for training and about 20% are used as hold-out set. This is the same set of dialogues as were used by the Bottleneck Simulator policy. No regularization is used.

We early stop on the squared error of the hold-out dataset w.r.t. Alexa user scores predicted by the reward model. At every turn in the dialogue, the corresponding policy picks the candidate response with the highest estimated score. As this policy was trained with an approximate state-action-value function, we call it *Q-Function Approx.*

We expect this policy to perform better compared to directly selecting the actions with the highest score under the reward model, because the learned policy is based on a deep neural network initialized using the crowdsourced labels.

Off-Policy REINFORCE Policy

The previous benchmark policies parametrized the estimated state-action-value function. Another way to parametrize the policy is as a discrete probability distribution over actions (candidate responses). In this case, the neural network outputs real-valued scores for each candidate response. These scores are then normalized through a softmax function, such that each candidate response is assigned a probability. This parametrization allows us to learn the policy directly from recorded dialogues through a set of methods known as *policy gradient* methods. This section describes one such approach.

Off-Policy Reinforcement Learning: We use a variant of the classical *REINFORCE* algorithm suitable for off-policy learning (Williams, 1992; Precup, 2000; Precup, Sutton, & Dasgupta, 2001).

As before, let $\{s_t^d, a_t^d, R^d\}_{d,t}$ be a set of observed (recorded) examples, where s_t^d is the dialogue history for dialogue d at time step (turn) t, a_t^d is the agent's action for dialogue d at time step (turn) t and R^d is the return for dialogue d. Let D be the number of observed dialogues and let T^d be the number of turns in dialogue d. Further, let θ_d be the parameters of the stochastic policy π_{θ_d} used during dialogue d. The algorithm updates the policy parameters θ by:

$$\Delta\theta \;\propto\; c_t^d \;\nabla_\theta \log \pi_\theta(a_t^d|s_t^d)\; R^d \quad \text{where } d \sim \text{Uniform}(1, D) \text{ and } t \sim \text{Uniform}(1, T^d), \tag{50}$$

where c_t^d is the importance weight ratio:

$$c_t^d \stackrel{\text{def}}{=} \frac{\prod_{t'=1}^{t} \pi_\theta(a_{t'}^d|h_{t'}^d)}{\prod_{t'=1}^{t} \pi_{\theta_d}(a_{t'}^d|h_{t'}^d)}. \tag{51}$$

This ratio corrects for the discrepancy between the learned policy π_θ and the policy under which the data was collected π_{θ_d} (sometimes referred to as the behaviour policy). It gives higher weights to examples with high probability under the learned policy and lower weights to examples with low probability under the learned reward function.

The importance ratio c_t^d is known to exhibit very high, possibly infinite, variance (Precup et al., 2001). Therefore, we truncate the products in the nominator and denominator to only include the current time step t:

$$c_{t,\text{trunc.}}^d \stackrel{\text{def}}{=} \frac{\pi_\theta(a_t^d|h_t^d)}{\pi_{\theta_d}(a_t^d|h_t^d)}. \tag{52}$$

This induces bias in the learning process, but also acts as a regularizer.

Reward Shaping: As mentioned before, one problem with the algorithm presented in eq. (50) is that it suffers from high variance (Precup et al., 2001). The algorithm uses

the return, observed only at the very end of an episode, to update the policy's action probabilities for all intermediate actions in the episode. With a small number of examples, the variance in the gradient estimator is overwhelming. This could easily lead the agent to over-estimate the utility of poor actions and, vice versa, to under-estimate the utility of good actions. One remedy for this problem is *reward shaping*, where the reward at each time step is estimated using an auxiliary function (Ng, Harada, & Russell, 1999). For our purpose, we propose a simple variant of reward shaping which takes into account the sentiment of the user. When the user responds with a negative sentiment (e.g. an angry comment), we will assume that the preceding action was highly inappropriate and assign it a reward of zero. Given a dialogue d, at each time t we assign reward r_t^d:

$$r_t^d \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if user utterance at time } t+1 \text{ has negative sentiment,} \\ \dfrac{R^d}{T^d} & \text{otherwise.} \end{cases} \tag{53}$$

With reward shaping and truncated importance weights, the learning update becomes:

$$\Delta\theta \propto c^d_{t,\text{trunc.}} \nabla_\theta \log \pi_\theta(a_t^d | s_t^d)\ r_t^d \quad \text{where } d \sim \text{Uniform}(1, D), t \sim \text{Uniform}(1, T^d), \tag{54}$$

Off-Policy Evaluation: To evaluate the policy, we estimate the expected return under the policy (Precup, 2000):

$$\text{R}_{\pi_\theta}[R] \ \approx\ \sum_{d,t} c^d_{t,\text{trunc.}}\ r_t^d. \tag{55}$$

Training: We initialize the policy with the parameters of the *Supervised* policy, and then train the parameters w.r.t. eq. (54) with stochastic gradient descent using Adam. We use a set of a few thousand dialogues recorded between users and a preliminary version of the system. The same set of recorded dialogues were used by the Bottleneck Simulator policy. About 60% of these examples are used for training, and about 20% are used for development and about 20% are used for testing. To reduce the risk of overfitting, we only train the parameters of the second last layer. We use a random grid search with different hyper-parameters, which include a temperature parameter and the learning rate. We select the hyper-parameters with the highest expected return on the development set. We call this policy *REINFORCE*.

Off-Policy REINFORCE with Learned Reward Function

Similar to the *Q-Function Approx.* policy, we may use the reward model for training with the off-policy REINFORCE algorithm. This section describes how we combine the two approaches.

Reward Shaping with Learned Reward Model: We use the reward model of the *Q-Function Approx.* policy to compute a new estimate for the reward at each time step in each dialogue:

$$r_t^d \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if user utterance at time } t+1 \text{ has negative sentiment,} \\ g_\phi(s_t, a_t) & \text{otherwise.} \end{cases} \tag{56}$$

This is substituted into eq. (54) for training and into eq. (55) for evaluation.

Training: As with the *REINFORCE* policy, we initialize this policy with the parameters of the *Supervised* policy, and then train the parameters w.r.t. eq. (54) with mini-batch stochastic gradient descent using Adam. We use the same set of dialogues and split as before. We use a random grid search with different hyper-parameters, As before, to reduce the risk of overfitting, we only train the parameters of the second last layer using this method. We select the hyper-parameters with the highest expected return on the development set. In this case, the expected return is computed according to the learned reward model.

This policy uses the learned reward model, which approximates the state-action-value function. This is analogous to the critic in an actor-critic architecture. Therefore, we call this policy *REINFORCE Critic*.

State Abstraction Policy

Finally, we describe an approach for learning a tabular state-action-value function based on state abstraction (Bean et al., 1987; Bertsekas & Castanon, 1989).

State Abstraction: We define the abstract policy state space to be the same as the set of abstract states Z used by the Bottleneck Simulator environment model described in Section 5.2:

$$Z = Z_{\text{Dialogue act}} \times Z_{\text{User sentiment}} \times Z_{\text{Generic user utterance}}. \tag{57}$$

This abstract state space contains a total of 60 discrete states. As with the Bottleneck Simulator environment model, the mapping $f_{s \to z}$ is used to map a dialogue history to its corresponding abstract state.

We define the abstract action space as the Cartesian product:

$$\mathcal{A} = \mathcal{A}_{\textit{Response model class}} \times \mathcal{A}_{\textit{Wh-question}} \times \mathcal{A}_{\textit{Generic response}}, \tag{58}$$

where $\mathcal{A}_{\textit{Response model class}}$ is a one-hot vector corresponding to one of the 13 response model classes[8] which generated the response, $\mathcal{A}_{\textit{Wh-question}} = \{True, False\}$ is a binary variable indicating whether or not the model response is a wh-question (e.g. a *what* or *why* question), and $\mathcal{A}_{\textit{Generic response}} = \{True, False\}$ is a binary variable indicating whether the response is generic and topic-independent (i.e. a response which only contains stop-words). The abstract action state space contains 52 abstract actions. A deterministic classifier is used to map an action (model response) to its corresponding abstract action.

In total, the tabular state-action-value function is parametrized by $60 \times 52 = 3120$ parameters.

Training: We train the policy with Q-learning on rollouts from the Bottleneck Simulator environment model. We use the same discount factor as the Bottleneck Simulator policy. We call this policy *State Abstraction*.

References

Abel, D., Hershkowitz, D. E., & Littman, M. L. (2016). Near Optimal Behavior via Approximate State Abstraction. In *ICML*.

8. Due to similarity between response models, some have been grouped together in the one-hot vector representation to reduce sparsity.

Asadi, K., Cater, E., Misra, D., & Littman, M. L. (2018). Equivalence Between Wasserstein and Value-Aware Loss for Model-based Reinforcement Learning. *arXiv preprint arXiv:1806.01265*, *1*.

Asri, L. E., He, J., & Suleman, K. (2016). A sequence-to-sequence model for user simulation in spoken dialogue systems. In *InterSpeech*.

Bansal, S., Calandra, R., Xiao, T., Levine, S., & Tomiin, C. J. (2017). Goal-driven dynamics learning via Bayesian optimization. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 5168–5173. IEEE.

Barreto, A. d. M. S., Beirigo, R. L., Pineau, J., & Precup, D. (2015). An Expectation-Maximization Algorithm to Compute a Stochastic Factorization From Data. In *IJCAI*.

Barreto, A. d. M. S., Beirigo, R. L., Pineau, J., & Precup, D. (2016). Incremental Stochastic Factorization for Online Reinforcement Learning. In *AAAI*.

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*(1-2), 41–77.

Bean, J. C., Birge, J. R., & Smith, R. L. (1987). Aggregation in dynamic programming. *Operations Research*, *35*(2), 215–220.

Bellemare, M., Veness, J., & Bowling, M. (2013). Bayesian learning of recursively factored environments. In *ICML*.

Bertsekas, D. P., & Castanon, D. A. (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, *34*(6), 589–598.

Bertsekas, D. P., & Tsitsiklis, J. N. (1995). Neuro-dynamic programming: an overview. In *IEEE Conference on Decision and Control, 1995*, Vol. 1, pp. 560–564. IEEE.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, *11*(1), 94.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *JMLR*, *3*(Oct), 213–231.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Brown, N., & Sandholm, T. (2017). Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. In *Science*.

Casanueva, I., Budzianowski, P., Su, P.-H., Ultes, S., Rojas-Barahona, L., Tseng, B.-H., & Gasic, M. (2018). Feudal reinforcement learning for dialogue management in large domains. In *NAACL-HLT*.

Chung, G. (2004). Developing a flexible spoken dialog system using simulation. In *ACL*.

Cuayahuitl, H., Renals, S., Lemon, O., & Shimodaira, H. (2005). Human-computer dialogue simulation using hidden Markov models. In *IEEE Workshop on ASRU*.

Cutler, M., Walsh, T. J., & How, J. P. (2015). Real-world reinforcement learning via multifidelity simulators. *IEEE Transactions on Robotics*, *31*(3), 655–671.

Das, A., Kottur, S., Moura, J. M., Lee, S., & Batra, D. (2017). Learning Cooperative Visual Dialog Agents with Deep Reinforcement Learning. In *International Conference on Computer Vision*.

Dean, T., Givan, R., & Leach, S. (1997). Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *UAI*.

Degris, T., Sigaud, O., & Wuillemin, P.-H. (2006). Learning the structure of factored markov decision processes in reinforcement learning problems. In *ICML*. ACM.

Deisenroth, M., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *ICML*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR*, *13*, 227–303.

Eckert, W., Levin, E., & Pieraccini, R. (1997). User modeling for spoken dialogue system evaluation. In *IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE.

Farahmand, A.-M., Barreto, A., & Nikovski, D. (2017). Value-Aware Loss Function for Model-based Reinforcement Learning. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Vol. 54.

Fatemi, M., Asri, L. E., Schulz, H., He, J., & Suleman, K. (2016). Policy networks with two-stage training for dialogue systems. In *SIGDIAL*.

Ferns, N., Castro, P. S., Precup, D., & Panangaden, P. (2006). Methods for Computing State Similarity in Markov Decision Processes. In *UAI*.

Ferns, N., Panangaden, P., & Precup, D. (2004). Metrics for finite Markov decision processes. In *UAI*.

Finn, C., & Levine, S. (2017). Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA)*. IEEE.

Francois-Lavet, V., Bengio, Y., Precup, D., & Pineau, J. (2019). Combined reinforcement learning via abstract representations. In *AAAI*.

Francois-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., Pineau, J., et al. (2018). An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, *11*(3-4), 219–354.

Gasic, M., & Young, S. (2011). Effective handling of dialogue state in the hidden information state POMDP-based dialogue manager. *ACM Transactions on Speech and Language Processing (TSLP)*, *7*(3), 4.

Georgila, K., Henderson, J., & Lemon, O. (2006). User Simulation for Spoken Dialogue Systems: Learning and Evaluation. In *Conference on Spoken Language Processing*.

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous Deep Q-Learning with Model-based Acceleration. In *ICML*.

Guo, X., Klinger, T., Rosenbaum, C., Bigus, J. P., et al. (2016). Learning to query, reason, and answer questions on ambiguous texts. In *ICLR*.

Hallak, A., Schnitzler, F., Mann, T., & Mannor, S. (2015). Off-policy model-based learning under unknown factored dynamics. In *ICML*.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *UAI*.

Heeman, P. A. (2009). Representing the reinforcement learning state in a negotiation dialogue. In *IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE.

Jiang, N., Kulesza, A., & Singh, S. (2015). Abstraction selection in model-based reinforcement learning. In *ICML*.

Jong, N. K., & Stone, P. (2005). State Abstraction Discovery from Irrelevant State Variables. In *IJCAI*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *JAIR*, *4*, 237–285.

Kansky, K., Silver, T., Mely, D. A., Eldawy, M., et al. (2017). Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics. In *ICLR*.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, *49*(2-3), 209–232.

Khouzaimi, H., Laroche, R., & Lefevre, F. (2017). Incremental human-machine dialogue simulation. In *Dialogues with Social Robots*, pp. 53–66. Springer.

Kingma, D., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *ICLR*.

Kreyssig, F., Casanueva, I., Budzianowski, P., & Gasic, M. (2018). Neural user simulation for corpus-based policy optimisation for spoken dialogue systems. *arXiv preprint arXiv:1805.06966*, *1*.

Kuvayev, L., & Sutton, R. (1997). Approximation in model-based learning. In *ICML*, Vol. 97.

Lee, S., & Eskenazi, M. (2012). POMDP-based let's go system for spoken dialog challenge. In *Spoken Language Technology Workshop (SLT), 2012 IEEE*, pp. 61–66. IEEE.

Lenz, I., Knepper, R. A., & Saxena, A. (2015). DeepMPC: Learning Deep Latent Features for Model Predictive Control. In *Robotics: Science and Systems*.

Levin, E., Pieraccini, R., & Eckert, W. (2000). A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on speech and audio processing*, *8*(1), 11–23.

Levine, S., et al. (2016). End-to-end training of deep visuomotor policies. *JMLR*, *17*(39), 1–40.

Lewis, M., Yarats, D., Dauphin, Y. N., Parikh, D., & Batra, D. (2017). Deal or No Deal? End-to-End Learning for Negotiation Dialogues. In *EMNLP*.

Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *ISAIM*.

Lillicrap, T. P., et al. (2016). Continuous control with deep reinforcement learning. In *ICLR*.

Liu, B., & Lane, I. (2017). Iterative Policy Learning in End-to-End Trainable Task-Oriented Neural Dialog Models. In *Proceedings of 2017 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.

Liu, C.-W., Lowe, R., Serban, I. V., Noseworthy, M., Charlin, L., & Pineau, J. (2016). How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation. In *EMNLP*.

Lopez-Cozar, R. (2016). Automatic creation of scenarios for evaluating spoken dialogue systems via user-simulation. In *Knowledge-Based Systems*. Elsevier.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., & Ma, T. (2019). Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *ICLR*.

McGovern, A., Sutton, R. S., & Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. In *Grace Hopper celebration of women in computing*, Vol. 1317.

Mnih, V., et al. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, *1*.

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

Narasimhan, K., Kulkarni, T., & Barzilay, R. (2015). Language Understanding for Text-based Games Using Deep Reinforcement Learning. In *EMNLP*.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, Vol. 99.

Oh, J., Guo, X., Lee, H., Lewis, R. L., & Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *NIPS*.

Osband, I., & Van Roy, B. (2014). Near-optimal Reinforcement Learning in Factored MDPs. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., & Weinberger, K. Q. (Eds.), *NIPS*.

Parr, R., & Russell, S. J. (1998a). Reinforcement learning with hierarchies of machines. In *NIPS*.

Parr, R. E., & Russell, S. (1998b). *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley, CA.

Peng, B., Li, X., Gao, J., Liu, J., & Wong, K.-F. (2018). Deep Dyna-Q: Integrating Planning for Task-Completion Dialogue Policy Learning. In *ACL*.

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

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*.

Petrik, M., & Subramanian, D. (2014). RAAM: The benefits of robustness in approximating aggregated MDPs in reinforcement learning. In *NIPS*.

Pieraccini, R., Suendermann, D., Dayanidhi, K., & Liscombe, J. (2009). Are we there yet? Research in commercial spoken dialog systems. In *Text, Speech and Dialogue*, pp. 3–13. Springer.

Png, S., Pineau, J., & Chaib-Draa, B. (2012). Building adaptive dialogue systems via bayes-adaptive POMDPs. *IEEE Journal of Selected Topics in Signal Processing*, *6*(8), 917–927.

Precup, D. (2000). Eligibility traces for off-policy policy evaluation. In *Computer Science Department Faculty Publication Series*.

Precup, D., Sutton, R. S., & Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In *ICML*.

Racaniere, S., Weber, T., Reichert, D., Buesing, L., et al. (2017). Imagination-Augmented Agents for Deep Reinforcement Learning. In *NIPS*.

Ram, A., Prasad, R., Khatri, C., Venkatesh, A., et al. (2017). Conversational AI: The Science Behind the Alexa Prize. In *Alexa Prize Proceedings*.

Ross, S. (2013). *Interactive learning for sequential decisions and predictions*. Ph.D. thesis, Carnegie Mellon University.

Ross, S., & Bagnell, J. A. (2012). Agnostic system identification for model-based reinforcement learning. In *ICML*.

Ross, S., & Pineau, J. (2008). Model-based Bayesian reinforcement learning in large structured domains. In *UAI*.

Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., & Young, S. (2007). Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *NAACL-HLT*.

Schulman, J., et al. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, *1*.

Serban, I. V., Sankar, C., Zhang, S., Lin, Z., Subramanian, S., Kim, T., Chandar, S., Ke, N. R., et al. (2017a). The Octopus Approach to the Alexa Competition: A Deep Ensemble-based Socialbot. In *Alexa Prize Proceedings*.

Serban, I. V., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subramanian, S., Kim, T., Pieper, M., Chandar, S., Ke, N. R., et al. (2017b). A Deep Reinforcement Learning Chatbot. *arXiv preprint arXiv:1709.02349*, *1*.

Shawar, B. A., & Atwell, E. (2007). Chatbots: are they really useful?. In *LDV Forum*, Vol. 22.

Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Silver, D., et al. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint arXiv:1712.01815*, *1*.

Singh, S., Litman, D., Kearns, M., & Walker, M. (2002). Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *JAIR*, *16*, 105–133.

Singh, S. P., Kearns, M. J., Litman, D. J., & Walker, M. A. (1999). Reinforcement Learning for Spoken Dialogue Systems. In *NIPS*.

Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., Taylor, P., Martin, R., Van Ess-Dykema, C., & Meteer, M. (2000). Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, *26*(3).

Strehl, A. L., Diuk, C., & Littman, M. L. (2007). Efficient structure learning in factored-state MDPs. In *AAAI*.

Su, P.-H., Gasic, M., Mrksic, N., Rojas-Barahona, L., et al. (2016). Continuously learning neural dialogue management. *arXiv preprint arXiv:1606.02689*, *1*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. No. 1 in 1. MIT Press Cambridge.

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.

Talvitie, E. (2015). Agnostic System Identification for Monte Carlo Planning. In *AAAI*, pp. 2986–2992.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58–68.

Traum, D., Marsella, S. C., Gratch, J., Lee, J., & Hartholt, A. (2008). Multi-party, multi-issue, multi-strategy negotiation for multi-modal virtual agents. In *International Workshop on Intelligent Virtual Agents*. Springer.

Tsybakov, A. B. (2009). *Introduction to nonparametric estimation*. Springer Series in Statistics. Springer, New York. Revised and extended from the 2004 French original. Translated by Vladimir Zaiats.

Van Roy, B. (2006). Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, *31*(2), 234–244.

Venkatraman, A., Capobianco, R., Pinto, L., Hebert, M., Nardi, D., & Bagnell, J. A. (2016). Improved learning of dynamics models for control. In *International Symposium on Experimental Robotics*. Springer.

Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *JAIR*, *12*, 387–416.

Wallace, R. S. (2009). The anatomy of ALICE. In *Parsing the Turing Test*. Springer.

Watter, M., et al. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*.

Wiering, M., Sałustowicz, R., & Schmidhuber, J. (1999). Reinforcement learning soccer teams with incomplete world models. *Autonomous Robots*, *7*(1), 77–88.

Wiering, M., & Schmidhuber, J. (1998). Efficient model-based exploration. In *Conference on Simulation of Adaptive Behavior*.

Williams, J. D., & Young, S. (2007a). Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language*, *21*(2), 393–422.

Williams, J. D., & Young, S. (2007b). Scaling POMDPs for spoken dialog management. *IEEE Transactions on Audio, Speech, and Language Processing*, *15*(7), 2116–2129.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4).

Wu, B., Feng, Y.-P., & Zheng, H.-Y. (2014). Model-based Bayesian Reinforcement Learning in Factored Markov Decision Process. *JCP*, *9*(4), 845–850.

Yu, Z., Xu, Z., Black, A. W., & Rudnicky, A. I. (2016). Strategy and Policy Learning for Non-Task-Oriented Conversational Systems. In *SIGDIAL*.

Zhao, T., & Eskenazi, M. (2016). Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In *SIGDIAL*.