Efficient Exploration for Dialog Policy Learning with Deep BBQ Networks & Replay Buffer Spiking

Zachary C. Lipton[1,2], Jianfeng Gao[2], Lihong Li[2], Xiujun Li[2], Faisal Ahmed[2], Li Deng[2]
University of California, San Diego[1]
Microsoft Research NExT, Redmond, WA[2]
`zlipton@cs.ucsd.edu`
`{jfgao, lihongli, xiul, fahmed, deng}@microsoft.com`

August 21, 2019

Abstract

When rewards are sparse and efficient exploration essential, deep Q-learning with ϵ-greedy exploration tends to fail. This poses problems for otherwise promising domains such as task-oriented dialog systems, where the primary reward signal, indicating successful completion, typically occurs only at the end of each episode but depends on the entire sequence of utterances. A poor agent encounters such successful dialogs rarely, and a random agent may never stumble upon a successful outcome in reasonable time. We present two techniques that significantly improve the efficiency of exploration for deep Q-learning agents in dialog systems. First, we demonstrate that exploration by Thompson sampling, using Monte Carlo samples from a Bayes-by-Backprop neural network, yields marked improvement over standard DQNs with Boltzmann or ϵ-greedy exploration. Second, we show that spiking the replay buffer with a small number of successes, as are easy to harvest for dialog tasks, can make Q-learning feasible when it might otherwise fail catastrophically.

1 Introduction

Enabled by improvements in automatic speech recognition and the ubiquity of instant messaging services, people increasingly interact with computers via automated dialog interfaces. Already, simple question answering (QA) bots are baked into products like Microsoft's Cortana, Google Now, Apple's Siri, and Amazon's Alexa. These QA bots typically carry out conversations consisting of a single exchange. On the other hand, we ultimately seek to develop general dialog agents, with the full breadth of capabilities exhibited by human conversants. In this work, as an intermediate step between the two, we consider task-oriented bots [Williams and Young, 2004], i.e., agents charged with some domain-specific goal, in our case, assisting a customer to book a movie.

While simple bots can be programmed with explicit policies, this proves untenable for these more complex settings for several reasons. First, it may be impossible or unreasonable to determine an optimal policy *a priori*. Second, the underlying dynamics of the problem may change over time, as say the database of available movies changes, or as the number of knowable attributes about each movie changes. Thus reinforcement learning (RL), in which policies are learned through interaction with an unknown environment, has emerged as a popular alternative [Levin et al., 1997, Walker, 2000, Lemon and Pietquin, 2007, Gašić et al., 2010]. In this work, inspired by recent breakthroughs on various domains such as Atari games [Mnih et al., 2013], board games [Silver et al., 2016] and some promising results on dialog [Fatemi et al., 2016, Cuayáhuitl, 2016, Williams and Zweig, 2016], we use deep reinforcement learning (DRL), an approach that weds the representational power of deep neural networks with the RL paradigm. However, unlike the simulation-based problems where DRL has made its greatest strides, human-interacting dialog systems incur significant real-world costs for failures. Additionally, task-oriented dialog is an especially hard problem because rewards come infrequently (at the end of dialogs). Consequently, a randomly initialized deep Q-network with ϵ-greedy exploration may never stumble upon a positive reward in the first place. Thus we hope to speed up learning by improving the efficiency of exploration and by introducing a simple mechanism to jump-start a deep Q-learner.

We express our problem as a partially observed Markov decision process (POMDP), a standard RL setting in which an agent interacts with an unknown environment. At each in a sequence of steps, the agent chooses an action given the current state of the environment. In turn, the agent receives some amount of reward and the environment transitions to a new state. If the dynamics of the environment are known a priori, this reduces to the task of *planning*. But absent such a model, an agent must learn the dynamics of its environment through exploration.

We consider deep Q-learning, a particularly successful approach, in which the agent learns a neural network model to predict the value (Q) of each (state, action) pair through temporal difference learning. Deep Q-learning has seen many recent breakthroughs, both video games [Mnih et al., 2013], board games [Silver et al., 2016], and dialog systems [Cuayáhuitl, 2016], owing largely to the use of deep neural networks to approximate value functions (deep Q-learning). To explore their environments, these systems typically explore via the ϵ-greedy heuristic. Given a state, a deep Q-network (DQN) predicts a value for each action. The agent chooses the action with the highest value with probability $1 - \epsilon$, and a random action with probability ϵ.

To be sure, ϵ-greedy has several advantages. In the limit, it results in infinite exploration. And when rewards are relatively frequent, such as points gained over the course of video game play, this strategy appears effective. However, in many reinforcement learning (RL) tasks, rewards are sparse. In these situations, a randomly exploring agent may never stumble upon a successful outcome (in feasible time).

We offer two solutions to improve exploration of Q-learners. First, we demonstrate that exploration by Thompson sampling, using Monte Carlo samples from a Bayes-by-Backprop neural network [Blundell et al., 2015], yields marked improvement over Boltzmann exploration and standard DQNs. At train time, we draw Monte Carlo samples from the *Bayes-by-Backprop Q-Network* (BBQN), both from the current network for the forward-pass and from the frozen *target network* [Mnih et al., 2013] to generate the targets. Second, we introduce *replay buffer spiking* (RBS), a technique in which we pre-fill the experience replay buffer with several episodes of experiences from a naive, but occasionally successful, rule-based agent. The presence of several positive rewards in the replay buffer prevents the Q network from converging on a local minima in which the predicted Q-value of all actions is 0 for all actions in all states. This trick proves essential for both BBQN and standard DQN learners, which otherwise achieve 0 reward.

We evaluate our agent on a movie-booking task-oriented dialog problem. Our agent must interact with a customer to book a movie. Success is assessed at the end of the dialog if (i) a movie has been booked and (ii) this movie satisfies the user. We benchmark the system using a user simulator, described in greater detail in 2. We evaluate our system and all baselines on a static problem and additionally present a domain extension experiment in which new actions and states become available over time.

Both experiments demonstrate that for this task, BBQNs outperform standard DQNs with both ϵ-greedy exploration, Boltzmann exploration, and a Thompson sampling approach based on bootstrapping due to Osband et al. [2016]. Additionally, we show that all systems only work given replay buffer spiking. Interestingly, while RBS proves es-

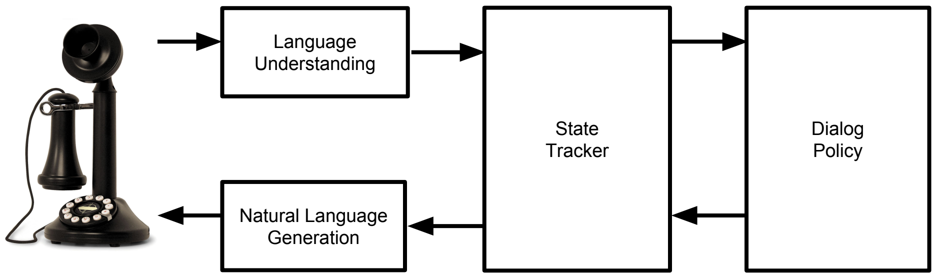


Figure 1: The basic components of a dialog system.

sential, the models are not especially sensitive to the number of episodes used to spike the buffers. Further experiments show that given a spiked replay buffer, uniformly random exploration confers no additional benefit either to the DQN or as supplement to the BBQN, possibly owing to the large action space.

2 Dialog Systems for Task Completion

We consider an automated agent tasked with helping a user to book a movie ticket. Over the course of several exchanges, an agent must gather information about the customer's desires, and ultimately book a suitable movie. The environment then assesses a binary outcome {success, failure} at the end of the conversation, based on (i) whether a movie is booked and (ii) whether the movie satisfies the user's constraints. In our experiments, success corresponds to a positive reward of 50, while failure corresponds to reward of 0.

Traditionally, dialog systems consist of pipelines resembling the one depicted in Figure 1 and detailed below. First, we gather a raw text utterance from a user, converting it to a *structured representation* via a *language understanding* module. Our structured representations follow the model of Schatzmann et al. [2007], which represents each utterance as a tuple containing a single *act* and a collection (possibly empty) of *(slot=value)* pairs, some of which some are *informed* while others are *requested*. Thus, some slots are paired with values while others are not. For example, the statement, "I'd like to see *Our Kind of Traitor* tonight in Seattle" maps to the structured representation *request(ticket, moviename=Our Kind of Traitor, starttime=tonight, city=Seattle).*

Next, the structured representation passes through a state tracker, which maintains a record of which slots have been filled. The state tracker also interacts with a database, providing the agent with information such as how many movies match the currently specified constraints. The state-tracker abstracts away information about the precise values filling the slots, enabling the policy module to act upon more generic, *de-lexicalized* representations, concerned with *intents* and *slots* but not values.

Finally, given a representation of the current conversation state provided by the state tracker, the policy module chooses among a set $\mathcal{A}$ of actions. For simplicity, we consider a set of 39 actions. These include the basic action such as *greeting(), thanks(), deny(), confirm_question, confirm_answer(), closing().* Additionally, we have two actions corresponding to each slot: one to inform its value and another to request it. While the policy chooses the slot to inform, the precise value informed is filled in by the state tracker. The pipeline then flows back towards the user. The chosen action passes to the state tracker, which fills in any vacant placeholders, yielding a structured representation such as *inform(theater=Cinemark Lincoln Square)*, which is then translated by a natural language generation component to a textual utterance, such as "This movie is playing tonight at Cinemark Lincoln Square".

When the user interacts via a voice interface, additional speech recognition and natural language generation components are added, but we omit them here to focus on the policy learning aspect of the pipeline. Many proposed systems differ from this pipeline, as by integrating several parts via more end-to-end machine learning agents. Nevertheless we find this description useful both for didactic purposes and contextualizing the present work.

Historically, dialog policies might be determined programatically. However, in complex or shifting domains, choosing an optimal policy *a priori* might be impossible. Thus, reinforcement learning has emerged as a popular alternative. A number of papers explore Gaussian processes for learning dialog policies [Gašić et al., 2010, Fatemi et al., 2016], using the GP-SARSA algorithm [Engel et al., 2005]. More recently, deep Q-learning, has shown promise, setting benchmarks on a number of tasks, and outperforming Gaussian processes on dialog policy learning [Fatemi et al., 2016]. However, given its nascence, research on the application of DRL to dialog policy learning remains insufficiently studied.

Learning a *task-oriented* dialog agent presents several distinct challenges for reinforcement learning. Rewards are especially sparse and unlike the video and board-game domains, sample complexity is of utmost importance. In a pre-release trial, each dialog may require the services of a paid product tester and in the wild, each failed dialog could exact both an opportunity cost and damage the reputation of the service. Thus we seek to expedite the learning process, especially in the early stages of learning.

Because training with live users could be costly, it's common to evaluate algorithms for learning dialog policies using a plausible user simulator. Of course, any simulator lacks the complexity of human conversants. Nevertheless, this test-bed provides a risk-free and reproducible environment, enabling research prior to a real-life deployment. For this work, we built a user simulator to mimic a goal-oriented customer seeking to purchase a movie ticket, modeled loosely on the agenda-based user simulator due to Schatzmann et al. [2007]. Against this virtual environment we perform all of our experiments [1].

3 Deep Q-Learning

We now introduce the fundamentals of deep Q-learning. An RL agent navigates a Markov decision process (MDP), interacting with its environment in a manner that unfolds over discreet time steps. At each step t, the agent observes the current state $s_t \in \mathcal{S}$, choosing some action $a_t \in \mathcal{A}$ according to a policy π. The agent then receives reward r_t and a new state s_{t+1}, and the cycle proceeds, until the episode terminates. Here, $\mathcal{S}$ represents the set of possible observations, $\mathcal{A}$ defines the space of possible actions and the policy $\pi : \mathcal{S} \to \mathcal{A}$ maps states onto actions. Here we assume actions to be discrete and $|\mathcal{A}|$ to be finite. Under a policy π and in state s the *value* of action a, we denote the expected cumulative discounted reward (also known as *return*) as:

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_t + \sum_{i=1}^{T} \gamma^i r_{t+i} \right] .$$

An optimal policy is whose Q-function uniformly dominates others; its value function, called optimal value function, is denoted Q^* [Sutton and Barto, 1998].

Given the optimal value function Q^*, at any time-step t, the optimal move is for the agent to choose action $a^* = \arg\max_a Q^*(s, a)$. Thus, learning an optimal policy can be reduced to learning the optimal value function. For toy problems, where an environment

[1] We will release code for both our models and the simulation system pubicly upon publication

can be fully explored, we can maintain an estimate of the Q function as a table of values, with rows indexing each state and columns for each action. In practice, however, the number of states may be intractably large, and the sample complexity of exploration can also grow at least linearly with the episode length T and the size of the action space $|\mathcal{A}|$. Thus, most practical reinforcement learners approximate the Q function by some parameterized model $Q(s, a; \theta)$, among which deep neural networks have become especially popular.

Moreover, the definition of return specifies a recursion: the value of the current state, action pair (s, a), depends upon the value of the successor state s_{t+1} and the action chosen in that state.

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')\,,$$

for some discount factor $\gamma \in (0, 1]$. For a fixed policy, the value function can be iteratively improved by a process called value iteration. We represent experiences as tuples (s_t, a_t, r_t, s_{t+1}). In Q-learning, we to improve the value function (and, in turn, the greedy policy) by minimizing the squared error between the current prediction and the one-step look-ahead prediction

$$L_i(\theta_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\rho(\cdot)}\left[(y_i - Q(s_t, a_t; \theta_i))^2\right] \tag{1}$$

for $y_i = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_t)$ and for $\rho(\cdot)$ denoting the joint distribution of experiences under the current policy. Traditionally, the Q-function is trained by stochastic approximation, estimating the loss on each experience as it is encountered, yielding the update:

$$\theta_{t+1} \leftarrow \theta_t - \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_t) - Q(s_t, a_t; \theta_t))\nabla Q(s_t, a_t; \theta_t). \tag{2}$$

Several widely used tricks improve the effectiveness of deep Q-learning. First, rather than training online, it's common to maintain a buffer of experiences, training on randomly selected mini-batches of experience [Lin, 1992, Mnih et al., 2013]. This technique, called *experience replay*, has proven effective and is believed to break up some of the correlations between the current estimate of the value function and the current policy [Mnih et al., 2015]. Second, it's been found effective to periodically cache the parameters θ^-, using these stale parameters to compute the training targets y_i. Other techniques such as double deep Q-learning [Van Hasselt et al., 2015] and prioritized experience replay [Schaul et al., 2016] have proven effective for learning the Q-function, *given* the replay experience. In contrast, our focus here is about generating "useful" replay experience—the issue of efficient exploration. For simplicity, we will use the basic DQN model and focus on the issue of exploration.

While standard deep Q-learning can efficiently estimate the values of policies, and improve upon them, the agent only observes the outcomes from actions chosen under its current or previous policies. In order to expose the agent to a richer set of experiences, one must employ a strategy for exploration. Most commonly in the DQN literature, researchers use the ϵ-greedy exploration heuristic. Recently other approaches have been considered. In this work, however, we seek to improve upon greedy exploration strategies by using uncertainty information to make more intelligent exploration choices.

4 Bayes-by-Backprop

In this section, we introduce Bayes-by-Backpropagation [Blundell et al., 2015], a method for extracting uncertainty information from neural networks by maintaining a probability distribution over the weights in the network. Without loss of generality, we confine the present discussion to multilayer perceptrons (MLPs), i.e., feedforward neural networks composed entirely of fully connected layers, without recurrent connections. A standard MLP, trained to perform regression, predicts $E[y|x]$, parameterized by weights $\boldsymbol{w} = \{W_l, b_l\}_{l=1}^{L}$. The MLP has the simple architecture:

$$y = W_L \cdot \sigma(W_{L-1} \cdot ... \cdot \ \sigma(W_2 \cdot \sigma(W_1 \cdot x + b_1) + b_2) + ... + b_{L-1}) + b_L$$

for a network with L layers (L-1 hidden) and nonlinear link function σ (commonly sigmoid, tanh, or rectified linear unit (ReLU). For a regression task, the topmost layer typically does not apply a nonlinearity.

In standard nerual network training, we learn the weights $\boldsymbol{w}$ given a dataset $\mathcal{D} = \{\boldsymbol{y}_i, \boldsymbol{x}_i\}_{i=1}^{N}$ by maximum likelihood estimation (MLE), using some variant of stochastic gradient descent, such as SGD, Adagrad, RMSprop, Adadelta, or Adam:

$$\begin{aligned} \boldsymbol{w}^{MLE} &= \arg\max_{\boldsymbol{w}} \ln p(\mathcal{D}|\boldsymbol{w}) \\ &= \arg\max_{\boldsymbol{w}} \sum_i \ln p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w}) \end{aligned} \tag{3}$$

Frequently, practitioners regularize models by placing priors on the parameters w. The resulting optimization seeks the maximum a posteriori (MAP) assignment of the weights $\boldsymbol{w^{MAP}}$, which can be expressed by Bayes rule as a function of both $p(\mathcal{D}|\boldsymbol{w})$ and $p(\boldsymbol{w})$, yielding ℓ_2^2 regularization if the prior is Gaussian, or ℓ_1 regularization with a Laplace prior:

$$\begin{aligned} \boldsymbol{w}^{MAP} &= \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\lceil) \\ &= \arg\max_{w} \ln p(\mathcal{D}|\boldsymbol{w}) P(\boldsymbol{w}) \\ &= \arg\max_{w} \ln p(\mathcal{D}|\boldsymbol{w}) + \ln p(\boldsymbol{w}) \end{aligned} \tag{4}$$

With a uniform prior, this simplifies to the maximum likelihood estimate.

Both maximum likelihood estimation and maximum a posteriori assignments produce a single value of $\boldsymbol{w}$, yielding a deterministic mapping function $f : \mathcal{X} \to \mathcal{Y}$. However, to enable efficient exploration, we prefer a model that can quantify uncertainty. Thus with a Bayesian treatment of neural networks, we learn a posterior distribution over the weights $p(\boldsymbol{w}|\mathcal{D})$ and not simply the mode of the posterior distribution. Problematically, $p(\boldsymbol{w}|\mathcal{D})$ may be intractable. The weights may be arbitrarily correlated and the joint distribution might be of arbitrary complexity, and thus difficult both to learn and to sample from.

To circumvent this problem, we use variational inference, approximating the potentially intractable posterior by a parametric distribution $q(\boldsymbol{w}|\theta)$. In this work, as in Blundell et al. [2015], we choose q to be Gaussian with diagonal covariance. Thus we sample each weight w_i from a univariate Gaussian distribution parameterized mean μ_i and standard deviation σ_i. To ensure that all σ_i remain strictly positive, we further parameterize the standard deviation with the softplus function $\sigma = \log 1 + \exp(\rho)$, giving variational parameters $\theta = \{(\mu_i, \rho_i)\}_{i=1}^{D}$ for D-dimensional weight vector $\boldsymbol{w}$.

We learn the parameters of this approximation by minimizing the Kullback Liebler

(KL) divergence between the variational approximation $q(\boldsymbol{w}|\theta)$ and the posterior $p(\boldsymbol{w}|\mathcal{D})$

$$\begin{aligned}\theta^* &= \arg\min_{\theta} \mathrm{KL}[q(\boldsymbol{w}|\theta)||p(\boldsymbol{w}|\mathcal{D})] \\ &= \arg\min_{\theta} \int_{-\infty}^{\infty} q(\boldsymbol{w}|\theta) \ln \frac{q(\boldsymbol{w}|\theta)}{p(\boldsymbol{w})p(\mathcal{D}|\boldsymbol{w})} \mathrm{d}\boldsymbol{w} \\ &= \arg\min_{\theta} \mathrm{KL}[q(\boldsymbol{w}|\theta)||p(w)] - \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\ln p(\mathcal{D}|\boldsymbol{w})].\end{aligned} \tag{5}$$

Thus we minimize the objective function $\mathcal{F}$, an expression which Hinton and Van Camp [1993] term the variational free energy:

$$\mathcal{F} = \mathrm{KL}[q(\boldsymbol{w}|\theta)||p(w)] - \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\ln p(\mathcal{D}|\boldsymbol{w})]$$

The term on the left term penalizes distance between the variational posterior $q(\boldsymbol{w}|\theta)$ and the prior $p(w)$. Specifically, the KL divergence measures the amount of information (measured in nats) that are lost when $p(\boldsymbol{w})$ is used to approximate $q(\boldsymbol{w}|\theta)$. Thus Hinton and Van Camp [1993] describe this term as a penalty on the description length of weights.

The rightmost term is simply the likelihood of the data given the weights, which under the assumption of a Gaussian error, corresponds to the expected square loss. When $q(\boldsymbol{w}|\theta)$ and $p(\boldsymbol{w})$ are both parameterized as univariate Gaussian distributions, their distance is minimized by setting $\mu_q = \mu_p$ for every setting of the variances, and by setting the standard deviations $\sigma_q = \sigma_p$ for any setting of the means μ_q and μ_p.

4.1 Minimising Loss By Stochastic Gradient Variational Bayes

Owing to this variational approach, we can learn a probabilistic model without sacrificing the efficiency of gradient-based training. That is, we learn the variational parameters θ by backpropagation, using a technique commonly termed the reparametrization trick Kingma and Welling [2013]. We want to differentiate the loss with respect to the variational parameters θ, but the loss depends upon the random vector $\boldsymbol{w} \sim q(\boldsymbol{w}|\theta)$. We can overcome this problem by expressing $\boldsymbol{w}$ as a deterministic function of θ, $g(\boldsymbol{\eta}, \theta)$, where $\boldsymbol{\eta}$ is a random vector. When we choose the function and noise distribution such that $p(\boldsymbol{\eta})d\boldsymbol{\eta} = q(\boldsymbol{w}|\theta)d\boldsymbol{w}$, we can express our optimization objective equivalently an expectation over η.

For any given value of η our loss is differentiable with respect to the variational parameters. We can then proceed with backpropagation, treating $\boldsymbol{\eta}$ as a noise input sampled for each batch. Thus, we minimize the loss by stochastic gradient descent, using a single Monte Carlo sample $\boldsymbol{\eta} \sim p(\boldsymbol{\eta})$ at each iteration. In our case, we take $\boldsymbol{\eta}$ to be a noise vector drawn from isotropic standard normal $\mathcal{N}(0, I)^D$. Thus $\boldsymbol{w} = g(\boldsymbol{\eta}, \theta) = \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \odot \boldsymbol{\eta}$, where $\odot$ is the element-wise product. For our work, the optimization objective becomes:

$$f(\boldsymbol{w}, \theta) = \log q(\boldsymbol{w}|\theta) - \log p(\boldsymbol{w}) - p(\mathcal{D}|\boldsymbol{w}) \tag{6}$$

5 Deep BBQ Learning

We now introduce the techniques used to train the proposed system.

5.1 Exploration by Thompson Sampling

To approximate the Q function, we use a Bayes by Backprop MLP as described above. When exploring the environment, we choose actions by Thompson sampling. To choose actions by Thompson sampling, we make one forward pass through the network with a single Monte Carlo sample of the weights $\boldsymbol{w} \sim q(\boldsymbol{w}|\theta)$, choosing whichever action, for that choice of the weights, corresponds to a higher value of the Q function. Additionally, we considered integrating ϵ-greedy approach by choosing via Thompson sampling probability $1-\epsilon$ and uniformly at random with probability ϵ. However, in our experiments, uniform random exploration conferred no supplementary benefit.

5.2 Training Procedure

When training, we sample weights both from the frozen target Q network and the current Q network. That is, when we freeze the network, we freeze all variational parameters of the target network. For each training batch, we use one Monte Carlo from the target network to construct the targets. We then take one Monte Carlo from the current Q network for the forward pass, applying a gradient update to the variational parameters.

5.3 Replay Buffer Spiking

Even using Thompson sampling to perform more intelligent exploration, the problem of reward sparsity remains. Any agent exploring completely at random may never stumble upon a first reward in time to guide further exploration. However, in the case of dialog we can produce a few successful dialogs manually, and mine these experiences to prefill the experience replay buffer. For these experiments, we construct a simple rule-based agent that, while far from optimal, achieves success sometimes and use it to spike the replay buffer.

In this work, we use a technique that we call replay buffer spiking (RBS), in which we pre-fill the replay buffer with some number of experiences collected from a rule-based agent. In our case the rule-based agent achieves success 18.3% of the time. We find that RBS is essential for both BBQN and DQN approaches to this task.

6 Experiments

To evaluate these techniques experimentally, we set up two problems. In each, the the agent interacts with the user simulator over 200 rounds. Each round consists of 50 simulated dialogs (episodes), followed by training for two epochs at the end of each round. In the first experiment, all slots are available starting from the very first dialog. In the second experiment, we evaluate the ability of each model to handle domain extension, specifically the introduction of new slots. Each time we add a new slot, we augment both the state space and action space. To accomplish this, we start out with only the essential slots: [*date, ticket, city, theater, starttime, moviename, numberofpeople, taskcomplete*]. With this initial set, we train for 40 training rounds. Then, every 10 rounds, we introduce a new slot in the following order: [*actor, critic_rating, description, genre, mpaa_rating, numberofkids, distance_constraints, other, price, state, theater_chain, video_format, zip*], thus expanding the possible user behavior. With each added slot the state space and action space grow accordingly.

6.1 Data

To represent the state of the environment at each turn, construct a 268 dimensional feature vector, consisting of the following:

- One hot representations of the *act* and *slot* corresponding to the current user action, with separate components for requested and informed slots.
- Corresponding represenations of the *act* and *slot* corresponding to the last agent action.
- A bag of *slots* corresponding to all previously filled slots over the course of the dialog history.
- Both a scalar and one hot representation of the current turn count.
- Counts representing the number of results from the knowledge base that match each presently filled in contraint (informed slot) as well as the intersection of all filled-in constraints.

For the domain extension experiments, the features corresponding to unseen slots all take value 0 until they are seen. Thus the domain could be extended by adding more features and corresponding weights, initializing the new weights to 0, a trick introduced by Lipton et al. [2015].

The agent has access to 39 actions. These correspond to the standard actions *greeting(), thanks(), confirm_question(), confirm_answer, deny(), closing()*. Additionally, we have one action to request and another to inform the value of each available slot. For the domain extension experiments, we mask the actions corresponding to unseen slots, revealing them only when those slots become available.

6.2 Training Details

Training in our experiments proceeds according to the following process. First we use a naive but occasionally successful rule-based agent for replay buffer spiking. While we present experiments comparing differrent amounts of spiking, final experiments use 100 dialogs to spike the replay buffer. Then, for each of 200 training rounds (i) the RL agent conducts 50 dialogs with the simulated user (ii) the agent freezes the target network parameters θ^{-}, then updates the Q function, training for 2 number of epochs, then re-freezing the target network and training for another two epochs.

Our motivations for proceeding in 50-dialog spurts rather than updating one minibatch per experience is two-fold. First, in a deployed dialog system, it may not be feasible to update the model in real-time. This method captures a more realistic periodic update. Second, our decision to allow the network to train more epochs per training round than is customary in DQN literature (usually one batch per experience) owes to the economics of dialog systems: Computational costs are negligible, while failed dialogs consume either human labor (in testing) or confer opportunity costs (in the wild). Thus, we are willing to suffer nearly arbitrary computational complexity to lower the sample complexity.

6.3 Baseline Methods

To demonstrate the efficacy of BBQ learning, we compare against a standard DQN. Additionally, we compare against Boltzmann exploration, a popular classical approach in which the probability of selecting each action in a given state is determined by a softmax function applied to the Q values. Finally, we compare against our implementation of the bootstrap method of Osband et al. [2016]. For the bootstrap experiments, we use 10 bootstrap heads, and assign each data point to each head with probability .5. We evaluate all four methods on both the full domain (static) learning problem and on the domain extension problem.

While we don't compare against Gaussian processes, we point to the recent work of Fatemi et al. [2016], which compares deep reinforcement learning methods (both policy gradient and Q-learning approaches) to GP-SARSA [Engel et al., 2005] on a similar dialog policy problem. The authors find that at present Q-learning approaches outperform GP-SARSA both with respect to both final performance, regret, and computational expense (by wall-clock). While we consider Gaussian processes to be a fascinating and evolving area, we do not consider them in this work.

6.4 Architecture Details

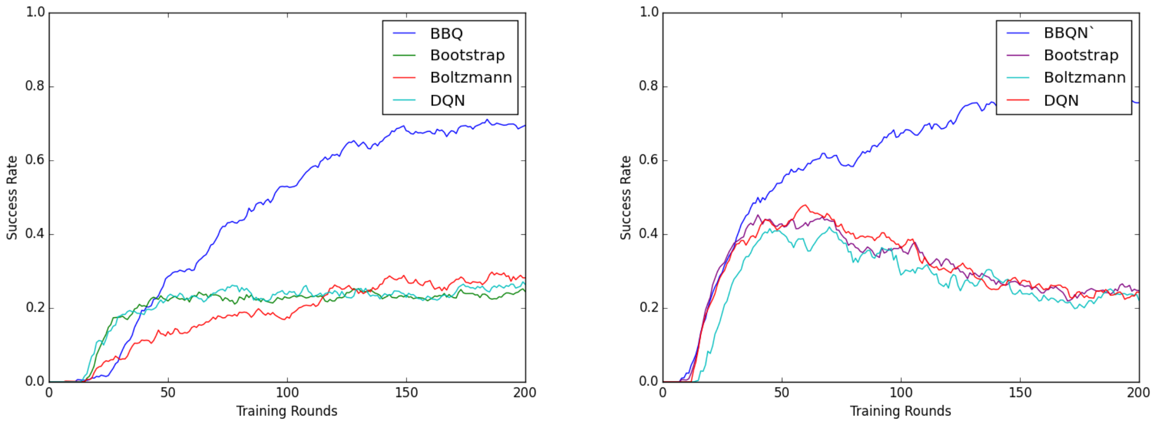
All models in this work use an MLP architecture with rectified linear units (ReLU) applied on hidden layers. Each network has 2 hidden layers, with 256 hidden nodes each. For ease of comparison we optimize each model using the Adam optimizer [Kingma and Ba, 2014], with a batch size of 32 and an initial learning rate of .001 as determined by a grid search. To avoid biasing the experiments towards our methods, we determined these common hyper-parameters using the DQN. Because the BBQN has built-in regularization, we equip all non-Bayesian neural networks with dropout regularization of .5, as this was shown to confer comparable predictive performance on holdout data by Blundell et al. [2015], a finding born out by our own experiments.

Each of our models had additional hyper=parameters. ϵ-greedy exploration requires an initial value of ϵ. Boltzmann exploration requires a softmax temperature. The bootstrapping-based method of Osband et al. [2016] requires both a number k of bootstrap heads and p a probability that each data point is assigned to each head. Our BBQ method requires that we determine the variance of the Gaussian prior distribution and the variance of the Gaussian error distribution.

6.5 Results

Final results demonstrate that the BBQN outperforms all baselines on both the full domain and domain extension problems both with respect to regret (Figure 2) and final performance of the trained model (Figure 3). Note that the domain extension problem grows more difficult every 10 epochs, so sustained performance corresponds to getting better while declining performance doesn't imply that the policy grows worse. Additionally, we present results showing the essential contribution of RBS (Figure 4). Interestingly, while RBS proved essential, but the BBQN and DQN both proved somewhat insensitive to the precise number of dialogs used to spike the buffer.

Finally, we considered that perhaps some promising trajectories might be ignored by the BBQN. Thus we constructed an experiment exploring via a hybridization of BBQN Thompson sampling with the ϵ-greedy approach. That is, with probability $1 - \epsilon$, the agent selects and action by Thompson sampling given one Monte Carlo sample from the BBQN and with probability ϵ the agent selects an action uniformly at random. Interestingly, the uniformly random exploration conferred no additional benefit. Further, we considered whether for the DQN, in the setting of replay buffer spiking, ϵ-greedy exploration performed any better than purely greedy exploration. Interestingly, non-zero values of ϵ resulted in higher regret and near-identical final performance of the trained models. Perhaps this owes to the relatively large state space of the dialog task.



(a) Full Domain (static) (b) Domain Extension

Figure 2: Training plots for the full domain (static) problem (a) and extended domain problem (b). Note that the latter grows more difficult every 10 epochs, so sustained performance corresponds to getting better while declining performance doesn't imply that the policy grows worse.

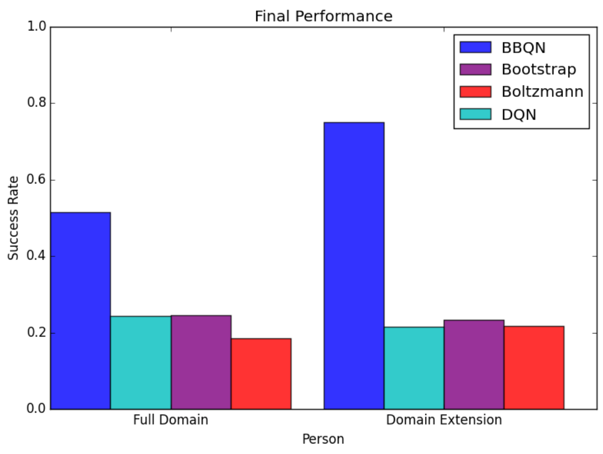


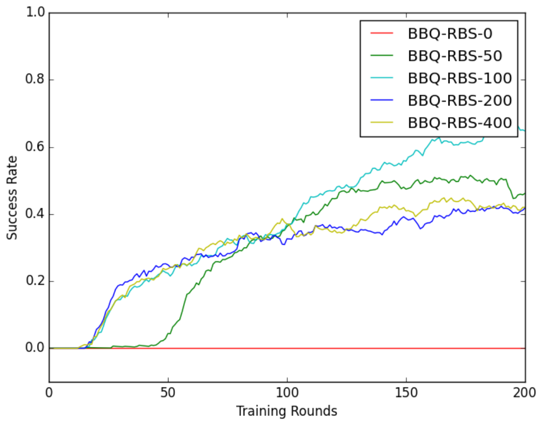
Figure 3: Final performance of the trained models on both the static and domain extension problems. In both cases the BBQN outperforms all baselines.

7 Related Work

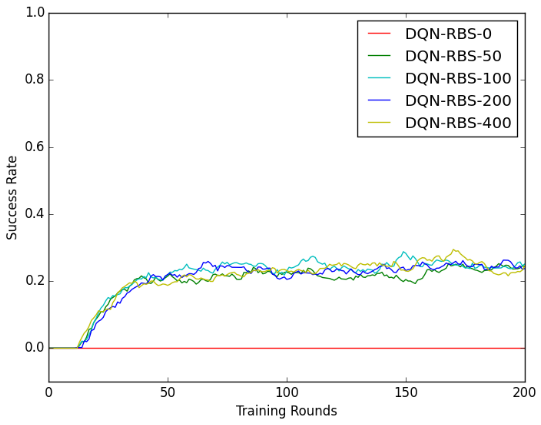
Our paper touches several areas of research, namely Bayesian neural networks, reinforcement learning with deep Q-networks, Thompson Sampling, and dialog systems, each of which boasts a rich history. While we cannot do justice to all, we briefly indicate the most influential and related work here.

7.1 Deep Q-Learning

This work builds on Q-learning [Watkins and Dayan, 1992], a popular method for model-free RL. For a broad resource on RL, we point to Sutton and Barto [1998]. While Q-learning has been used with neural networks for decades, the method has seen recent resurgence as researchers have wedded the technique with modern deep learning techniques to approximate the Q-function. Namely, Mnih et al. [2013, 2015] first achieved super-human performance on a wide range of Atari games, using deep convolutional neural networks and incorporating previously described tricks such as experience replay [Lin,

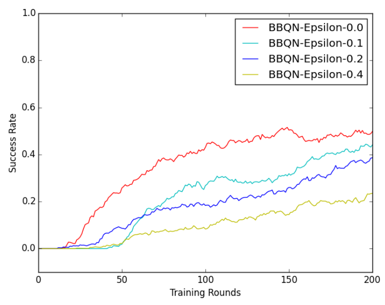


(a) BBQN

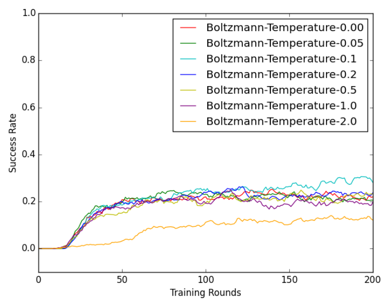


(b) DQN

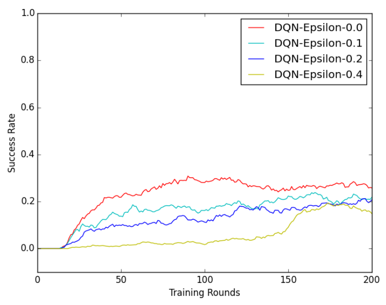
Figure 4: For both the BBQN and DQN, replay buffer spiking proves essential. Otherwise, the network converges to the local optimum of assuming all actions have value 0. Interestingly, the networks do not appear sensitive to the number of warmstart dialogs, performing well with as few as 50 (corresponding to ≈ 10 successes).



(a) BBQN & ϵ-Greedy



(b) DQN & Boltzmann Exploration



(c) DQN & ϵ-greedy

Figure 5: Interestingly, no amount of uniformly random exploration conferred a benefit for BBQ network. Similarly, Boltzmann exploration didn't appear helpful. Plots indicate the initial values of the temperature and ϵ respectively, both of which are then attenuated over the course of training.

1992]. Several follow-up works [Schaul et al., 2016, Van Hasselt et al., 2015, Wang et al., 2015] improve on these results in various ways.

7.2 Efficient Exploration

Efficient exploration has been one of the defining challenges in reinforcement learning. While provably efficient exploration strategies are known for problems with finite states/actions or problems with "nice" structures [Kakade, 2003, Asmuth et al., 2009, Jaksch et al., 2010, Li et al., 2011, Osband et al., 2013], not much is known for the general case, especially when function approximation is involved.

The first approaches to deep Q learning rely upon the ϵ-greedy exploration heuristic [Mnih et al., 2015]. More recently, Stadie et al. [2015] and Houthooft et al. [2016] introduced approaches to encourage exploration by perturbing the reward function. In these approaches, one model is trained to predict the dynamics of the environment, actions which lead to surprising results (not predicted by the dynamics model) are given a bonus reward. However, this these approaches are somewhat orthogonal to our own approach as our method could be applied for any fixed reward function. Osband et al. [2016] attempts to mine uncertainty information by training a neural network with multiple output *heads*. Each head produces a value for each action and is associated with a distinct subset of the data. While they show improved results on some but not all Atari games, this approach does not appear to confer an advantage over standard DQNs on our problem.

The Bayes by backpropagation (BB) method we build on was first introduced by Blundell et al. [2015], and employs the Stochastic Gradient Variational Bayes approach and reparametrization trick popularized by Kingma and Welling [2013]. The method follows many years of interest in variational treatments of neural networks, notably Hinton and Van Camp [1993] and Graves [2011]. The authors evaluate using Bayes-by-Backpropagation with Thompson sampling for a contextual bandit problem, but do not evaluate adapting the method for Q-learning.

While efficient exploration represents a new and burgeoning area of research in deep reinforcement learning, the topic has been studied extensively with classical reinforcement learning methods. Chapelle and Li [2011] present a thorough empirical examination of Thompson sampling, one of the oldest heuristics for exploration [Thompson, 1933], for contextual bandits; the same exploration strategy is also found helpful in reinforcement learning with finite MDPs [Strens, 2000, Osband et al., 2013]. In contrast, this work extends Thompson sampling to reinforcement learning with function approximation.

7.3 Dialog Systems

Our work builds on a rich body of work in developing task oriented dialog systems [Williams and Young, 2004, Gašić et al., 2010, Wen et al., 2016] and RL for learning dialog policies [Levin et al., 1997, Singh et al., Walker, 2000, Lemon and Pietquin, 2007, Gašić et al., 2010]. We also follow the example of Cuayáhuitl [2016], Fatemi et al. [2016] both of whom consider deep reinforcement learning for dialog policies. Additionally, our domain-extension experiments are inspired by Gašic et al. [2014]. The agenda-based user simulator against which we construct all of our experiments was inspired by the work of Schatzmann et al. [2007].

8 Discussion

Our results suggest that for learning dialog policies BBQ networks explore with far greater efficiency than traditional epsilon greedy approaches, Boltzmann explorations and the recently published Bootstrap techniques of Osband et al. [2016]. The results are similarly stark for both static and domain extension experiments.

We see several promising paths for future work. First, given the substantial improvements of BBQ over other exploration strategies, we would like to re-implement this work on the standard deep reinforcement learning benchmark tasks (Atari, etc.) to see if it confers a similarly stark improvement. Additionally, we would like to combine BBQ learning with other, orthogonal approaches, which work by perturbing the reward function to add a bonus for uncovering surprising transitions, i.e., state transitions given low probability by a dynamics model Stadie et al. [2015], Houthooft et al. [2016]. Conceivably these approaches could be complementary. Our BBQ net addresses uncertainty in the Q-value given the current policy, whereas curiosity address uncertainty over what else might by possible in policy space, that might never be tried under the current policy. We anticipate a potential synergistic effect of combining the two.

References

John Asmuth, Lihong Li, Michael L. Littman, Ali Nouri, and David Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*, pages 19–26, 2009.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011.

Heriberto Cuayáhuitl. Simpleds: A simple deep reinforcement learning dialogue system. *arXiv preprint arXiv:1601.04574*, 2016.

Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208. ACM, 2005.

Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. Policy networks with two-stage training for dialogue systems. *arXiv preprint arXiv:1606.03152*, 2016.

M Gašić, F Jurčíček, Simon Keizer, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. Gaussian processes for fast policy optimisation of pomdp-based dialogue managers. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 201–204. Association for Computational Linguistics, 2010.

Milica Gašic, Dongho Kim, Pirros Tsiakoulis, Catherine Breslin, Matthew Henderson, Martin Szummer, Blaise Thomson, and Steve Young. Incremental on-line adaptation of pomdp-based dialogue managers to extended domains. *In Proceedings on InterSpeech*, 2014.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *arXiv preprint arXiv:1605.09674*, 2016.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

Sham Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, Gatsby Computational Neuroscience Unit, University College London, UK, 2003.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Olivier Lemon and Olivier Pietquin. Machine learning for spoken dialogue systems. In *European Conference on Speech Communication and Technologies (Interspeech'07)*, pages 2685–2688, 2007.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. Learning dialogue strategies within the markov decision process framework. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, pages 72–79. IEEE, 1997.

Lihong Li, Michael L. Littman, Thomas J. Walsh, and Alexander L. Strehl. Knows what it knows: A framework for self-aware learning. *Machine Learning*, 82(3):399–443, 2011.

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

Zachary C Lipton, Sharad Vikram, and Julian McAuley. Capturing meaning in product reviews with character-level generative text models. *arXiv preprint arXiv:1511.03683*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Ian Osband, Dan Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3003–3011, 2013.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016.

Jost Schatzmann, Blaise Thomson, and Steve Young. Statistical user simulation with a hidden agenda. *Proc SIGDial, Antwerp*, 273282(9), 2007.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *Proceedings of the 4th International Conference on Learning Representations*, 2016. arXiv preprint arXiv:1511.05952.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. Reinforcement learning for spoken dialogue systems.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Malcolm J. A. Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 943–950, 2000.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2015.

Marilyn A. Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12: 387–416, 2000.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

Jason D Williams and Steve J Young. Characterizing task-oriented dialog using a simulated ASR channel. In *INTERSPEECH 2004 - ICSLP, 8th International Conference on Spoken Language Processing*, 2004.

Jason D. Williams and Geoffrey Zweig. End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. *CoRR*, abs/1606.01269, 2016. URL `http://arxiv.org/abs/1606.01269`.