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

Article · August 2016

CITATIONS
29

READS
342

6 authors, including:

Zachary Chase Lipton
Carnegie Mellon University
128 PUBLICATIONS **5,561** CITATIONS
SEE PROFILE

Jianfeng Gao
Chinese Academy of Sciences
413 PUBLICATIONS **23,292** CITATIONS
SEE PROFILE

Lihong Li
125 PUBLICATIONS **7,553** CITATIONS
SEE PROFILE

Faisal Ahmed
University of California, Irvine
8 PUBLICATIONS **399** CITATIONS
SEE PROFILE

Efficient Exploration for Dialogue Policy Learning with 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

Abstract

When rewards are sparse and action spaces large, Q-learning with ϵ-greedy exploration can be inefficient. This poses problems for otherwise promising applications such as task-oriented dialogue systems, where the primary reward signal, indicating successful completion of a task, requires a complex sequence of appropriate actions. Under these circumstances, a randomly exploring agent might 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 dialogue systems. First, we introduce an exploration technique based on Thompson sampling, drawing Monte Carlo samples from a Bayes-by-backprop neural network, demonstrating marked improvement over common approaches such as ϵ-greedy and Boltzmann exploration. Second, we show that spiking the replay buffer with experiences from a small number of successful episodes, as are easy to harvest for dialogue tasks, can make Q-learning feasible when it might otherwise fail.

1 Introduction

Increasingly, we interact with computers via natural-language dialogue interfaces. Simple question answering (QA) bots already serve millions of users through Amazon's Alexa, Apple's Siri, Google's Now, and Microsoft's Cortana. While these bots typically carry out single-exchange conversations, we aspire to develop more general dialogue agents, with the full breadth of capabilities exhibited by human interlocutors. 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 movie tickets.

Simple bots can be programmed with explicit policies but this approach can break down for several reasons. First, it may be impossible to determine an acceptable policy *a priori*. Second, the underlying dynamics of the problem may change over time, as say the database of available movies changes. Thus, reinforcement learning (RL), in which policies are learned through interaction with an unknown and possibly changing environment, has emerged as a popular alternative [Singh et al., 2000, Gašić et al., 2010, Fatemi et al., 2016]. Inspired by recent RL breakthroughs in the Atari games [Mnih et al., 2015] and board games [Silver et al., 2016] domains, we use deep reinforcement learning (DRL), an approach that weds the representational power of deep neural networks with the RL paradigm. To explore their environments, DRL systems typically employ the ϵ-greedy heuristic. And when rewards are relatively frequent, such as points gained over the course of video game play, this strategy appears effective.

However, for dialogue systems, the primary reward signal is sparse and the action space is large, conditions under which ϵ-greedy fails. For our problem, a randomly exploring Q-learner may never

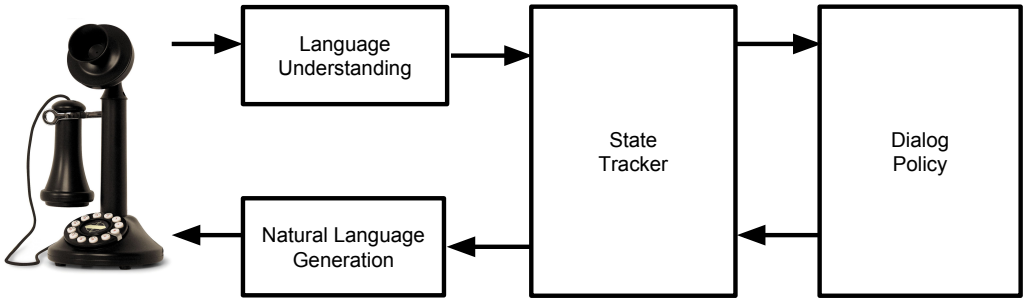


Figure 1: The basic components of a dialogue system.

stumble upon a successful dialogue. Moreover, unlike the video game and board game domains, human-interacting dialogue systems incur significant real-world costs for failures. 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 offer two solutions to improve the exploration of Q-learners. First, we introduce the Bayes-by-backprop Q-network (BBQN), a technique for exploring via Thompson sampling, drawing Monte Carlo samples from a Bayes-by-backprop neural network [Blundell et al., 2015]. Second, we introduce *replay buffer spiking* (RBS), a simple technique in which we pre-fill the experience replay buffer with several episodes of experiences from a naive, but occasionally successful, rule-based agent. This proves essential for both BBQN and standard deep Q-networks (DQNs), which otherwise achieve 0 successful dialogues.

We evaluate our agents on two movie-booking dialogue problems, bench-marking the system using an agenda-based user simulator similar to Schatzmann et al. [2007]. In the first problem, the environment remains fixed. The second problem explores domain-extension, a setting in which new attributes of films become available over time. Both experiments demonstrate that BBQNs outperform standard DQNs with either ϵ-greedy exploration, Boltzmann exploration, or the bootstrap approach due to Osband et al. [2016].

2 Dialogue Systems for Task Completion

We consider a dialog system for helping a user to book movie tickets. Over the course of several exchanges, the agent gathers information about the customer's desires and ultimately books a suitable movie. The environment then assesses a binary outcome (success or 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 reward of 40, failure to a reward of -10, and we assess a per turn penalty of -1 to encourage pithy exchanges.

Typically, dialogue pipelines resemble Figure 1 and contain the following components: First, we convert raw text from a user, to a structured representation via a *language understanding* module. Following Schatzmann et al. [2007], we represent each utterance as a single *act* and a (possibly empty) collection of *(slot=value)* pairs, some of which are *informed* while others are *requested* (value omitted). For example, the utterance, "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 current constraints. The state-tracker *de-lexicalizes* the utterances, enabling the policy module to act upon more generic states, concerned with *acts* and *slots* but not precise values.

Given this abstract representation of the dialogue state, the policy module chooses among a set $\mathcal{A}$ of *actions*, which consist of full structured utterances (not *acts*. In the current movie-booking task, for simplicity, we consider a set of 39 actions. These include the basic actions 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.

The pipeline then flows back towards the user. Any slots informed by the policy are then filled in by the state tracker. 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".

Many proposed systems in the literature differ from this pipeline, as by integrating several parts via more end-to-end machine learning agents or including speech-to-text components. Nevertheless, we find this setup useful for investigating the fundamental policy learning problems.

Because training with live users could be costly, it's common to evaluate algorithms for learning dialogue policies using a plausible user simulator. Of course, any simulator lacks the complexity of a human interlocutor. 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

An RL agent navigates a Markov decision process (MDP), interacting with its environment over a sequence of discrete steps. At each step t, the agent observes the current state $s_t \in \mathcal{S}$, and chooses some action $a_t \in \mathcal{A}$ according to a policy π. The agent then receives reward r_t and observes new state s_{t+1}, continuing the cycle until the episode terminates. Here, $\mathcal{S}$ represents the set of all possible states, $\mathcal{A}$ defines the space of possible actions and the policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ maps states onto actions. In this work, 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 is the expected cumulative discounted reward (also known as *return*):

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{i=0}^{T} \gamma^i r_{t+i} | s_t = s, a_t = a\right]$$

where γ is a discount factor. An optimal policy is one whose Q-function uniformly dominates others. Its value function, called the *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. In practice, the number of states may be intractably large, and the sample complexity of exploration can grow at least linearly with the number of states $|S|$ 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.

For a fixed policy, the value function can be iteratively improved by approximate value iteration. We represent experiences as tuples (s_t, a_t, r_t, s_{t+1}). In Q-learning, we aim 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

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

for $y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_t)$ 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:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(y_t - Q(s_t,a_t;\theta_t))\nabla Q(s_t,a_t;\theta_t). \tag{2}$$

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., 2015]. This technique, called *experience replay*, has proven

[1] We will release code for our simulator upon publication.

effective and is believed to break up the tight coupling between the observed states and the current policy [Mnih et al., 2015]. Second, it's common to periodically cache the parameters θ^-, using these stale parameters to compute the training targets y_t.

Other techniques such as double deep Q-learning [Van Hasselt et al., 2015] and prioritized experience replay [Schaul et al., 2016] appear effective for learning the Q-function. For simplicity and because these techniques are straightforward to combine with ours, we build on the basic DQN model and focus on the issue of exploration.

In order to expose the agent to a rich set of experiences, one must employ a strategy for exploration. Most commonly in the DQN literature, researchers use the ϵ-greedy exploration heuristic. Recently, several papers have considered other approaches. In this work, we seek to improve upon greedy exploration strategies by using uncertainty information (in the predicted Q values) to make more intelligent exploration choices.

4 Bayes-by-Backprop

We now introduce Bayes-by-backprop [Blundell et al., 2015], a method for extracting uncertainty information from neural networks by maintaining a probability distribution over the weights in the network. 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 for regression models $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w})$, parameterized by weights $\boldsymbol{w} = \{W_l, b_l\}_{l=1}^{L}$. MLPs have the simple architecture:

$$\hat{\boldsymbol{y}} = W_L \cdot \phi(W_{L-1} \cdot ... \cdot \ \phi(W_1 \cdot \boldsymbol{x} + b_1) + ... + b_{L-1}) + b_L$$

for a network with L layers ($L-1$ hidden) and activation function ϕ (commonly sigmoid, tanh, or rectified linear unit (ReLU).

In standard neural network training, we learn the weights $\boldsymbol{w}$ given a dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$ by maximum likelihood estimation (MLE), using some variant of stochastic gradient descent: $\boldsymbol{w}^{MLE} = \arg\max_{\boldsymbol{w}} \log p(\mathcal{D}|\boldsymbol{w})$. Frequently, we regularize models by placing priors on the parameters $\boldsymbol{w}$. The resulting optimization seeks the maximum a posteriori (MAP) assignment $\boldsymbol{w}^{\boldsymbol{MAP}} = \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{D})$.

Both MLE and MAP assignments produce point estimates of $\boldsymbol{w}$, and thus capture only the mode of the predictive distribution, But to enable efficient exploration, we prefer a model that can quantify uncertainty. Thus, we consider a Bayesian treatment of neural networks, learning a full posterior distribution $p(\boldsymbol{w}|\mathcal{D})$.

Problematically, $p(\boldsymbol{w}|\mathcal{D})$ may be intractable. So we approximate the potentially intractable posterior by a variational distribution $q(\boldsymbol{w}|\theta)$. In this work, we choose q to be Gaussian with diagonal covariance. Thus, we sample each weight w_i from a univariate Gaussian distribution parameterized by mean μ_i and standard deviation σ_i. To ensure that all σ_i remain strictly positive, we parameterize σ_i by the softplus function $\sigma_i = \log(1 + \exp(\rho_i))$, giving variational parameters $\theta = \{(\mu_i, \rho_i)\}_{i=1}^{D}$ for D-dimensional weight vector $\boldsymbol{w}$.

Note that the true posterior is both multi-modal (owing to symmetry among the nodes) and intractable. There is no reason to believe that the true posterior exhibits conditional independence between every pair of two weights regardless of the values taken by the others. So this is only an approximation in a very narrow sense. Nonetheless, it proves useful in practice.

We learn these parameters by minimizing the Kullback-Liebler (KL) divergence between the variational approximation $q(\boldsymbol{w}|\theta)$ and the posterior $p(\boldsymbol{w}|\mathcal{D})$

$$\begin{aligned}
\theta^* &= \arg\min_{\theta} \mathrm{KL}[q(\boldsymbol{w}|\theta)||p(\boldsymbol{w}|\mathcal{D})] \\
&= \arg\min_{\theta} \int q(\boldsymbol{w}|\theta) \log \frac{q(\boldsymbol{w}|\theta)}{p(\boldsymbol{w})p(\mathcal{D}|\boldsymbol{w})} \mathrm{d}\boldsymbol{w} \\
&= \arg\min_{\theta} \mathrm{KL}[q(\boldsymbol{w}|\theta)||p(w)] - \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\log p(\mathcal{D}|\boldsymbol{w})].
\end{aligned} \tag{3}$$

The expression minimized is termed by Hinton and Van Camp [1993] the *variational free energy*. Assuming Gaussian error, the rightmost term is simply the expected square loss. Sampling from q, our cost function is $f(\mathcal{D}, \theta) = \log q(\boldsymbol{w}|\theta) - \log p(\boldsymbol{w}) - \log p(\mathcal{D}|\boldsymbol{w})$.

We can learn the variational parameters θ by gradient descent, using the reparametrization trick popularized by Kingma and Welling [2013]. In short, 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 g and noise distribution $p(\boldsymbol{\eta})$ such that $p(\boldsymbol{\eta})d\boldsymbol{\eta} = q(\boldsymbol{w}|\theta)d\boldsymbol{w}$, we can express our optimization objective equivalently as an expectation over η. In our case, we take $\boldsymbol{\eta}$ to be a noise vector drawn from D-dimensional standard normal $\mathcal{N}(\mathbf{0}, I)^D$. Thus $\boldsymbol{w} = g(\boldsymbol{\eta}, \theta) = \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \odot \boldsymbol{\eta}$, where $\odot$ is the element-wise product.

5 Deep BBQ-Learning

We now introduce the techniques used to train the proposed system. 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. Precisely, 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 the highest value of the Q-function. We also considered integrating ϵ-greedy approach, exploring by Thompson sampling with probability $1 - \epsilon$ and uniformly at random with probability ϵ. But empirically, uniform random exploration conferred no supplementary benefit.

We initialize the variational parameters to match the prior. So $\boldsymbol{\mu}$ is initialized to the zero vector $\mathbf{0}$ and $\boldsymbol{\rho}$ to match the variance of the prior. Note that unlike with conventional neural networks, we need not assign the weights randomly as sampling breaks symmetry. As a consequence of this initialization, from the outset, the agent explores uniformly at random. Over the course of training, as the buffer fills, the mean squared error starts to dominate the objective function and the variational distribution moves further from the prior.

When we freeze the network, we freeze all variational parameters of the target network. Then, during training, for each mini-batch, we draw one Monte Carlo sample of the weights from the frozen target Q-network's variational distribution to construct the targets. We then draw one Monte Carlo sample from the current Q-network's variational distribution for the forward pass. On the backwards pass, we apply a gradient update to the current variational parameters. Note that if we draw one sample per example, random number generation becomes the rate-limiting operation during training. By sampling once per mini-batch, we amortize this computational expense. Using this approach, BBQ and DQN training speeds are roughly equivalent.

While Thompson sampling is a useful strategy for exploration, it does not necessarily handle reward sparsity well *at the beginning of learning*. Any agent exploring completely at random may never stumble upon a first reward in time to guide further exploration. However, in the case of dialogue we can produce a few successful dialogues manually, using these experiences to pre-fill the experience replay buffer. For these experiments, we construct a simple rule-based agent that, while sub-optimal (18.3% success rate), achieves success sometimes. We then harvest experiences from some number of rule-based dialogues, adding them to the replay buffer. We call this trick *replay buffer spiking (RBS)*. We find that, on our task, RBS is essential for both BBQN and DQN approaches.

6 Experiments

We evaluate our methods on two problems. In the first, the agent interacts with the user simulator over 400 rounds. Each round consists of 50 simulated dialogues, followed by 2 two epochs of training. All slots are available starting from the very first dialog. In the second experiment, we test each model's ability to adapt to domain extension periodically introducing new slots. Each time we add a new slot, we augment both the state space and action space. We start out with only the essential slots: [*date, ticket, city, theater, starttime, moviename, numberofpeople, taskcomplete*] and train for 40 training rounds up front. Then, every 10 rounds, we introduce a new slot in a fixed but arbitrary order. With each added slot the state space and action space (for both user and agent) grow accordingly. This experiment terminates after 200 rounds.

To represent the state of the environment at each turn, we construct a 268 dimensional feature vector, consisting of the following: (i) One hot representations of the *act* and *slot* corresponding to the current user action, with separate components for requested and informed slots. (ii) Corresponding representations of the *act* and *slot* corresponding to the last agent action. (iii) A bag of *slots*

corresponding to all previously filled slots over the course of the dialog history. (iv) Both a scalar and one hot representation of the current turn count. (v) Counts representing the number of results from the knowledge base that match each presently filled in constraint (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 or $\mu_i = 0, \sigma_i = \sigma_{prior}$ for BBQ, a trick due to Lipton et al. [2015].

6.1 Training Details:

To train our models, we first use a naive but occasionally successful rule-based agent for replay buffer spiking. Final experiments use 100 dialogs to spike the replay buffer. We note that experiments showed models to be insensitive to the precise number. After each round of 50 simulated dialogues, the agent freezes the target network parameters θ^-, and then updates the Q function, training for 2 epochs, then re-freezes and trains for another 2 epochs.

Our motivations for proceeding in 50-dialog spurts rather than updating one mini-batch per experience is two-fold. First, in a deployed dialog system, real-time updates might not be realistic. Second, we train for more batches per new experience than is customary in DQN literature owing to the economic considerations: Computational costs are negligible, while failed dialogues consume either human labor (in testing) or confer opportunity costs (in the wild).

6.2 Baseline Methods:

To demonstrate the efficacy of BBQ learning, we compare against a standard DQN. Additionally, we compare against Boltzmann exploration, an approach in which the probability of selecting any action in a given state is determined by a softmax function applied to the predicted Q values. Here affinity for exploration vs exploration is parameterized by the Boltzmann *temperature*. We also compare to 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.

We don't compare against Gaussian processes, but point to the recent work of Fatemi et al. [2016], which compares DRL 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 with respect to both final performance, regret, and computational expense (by wall-clock).

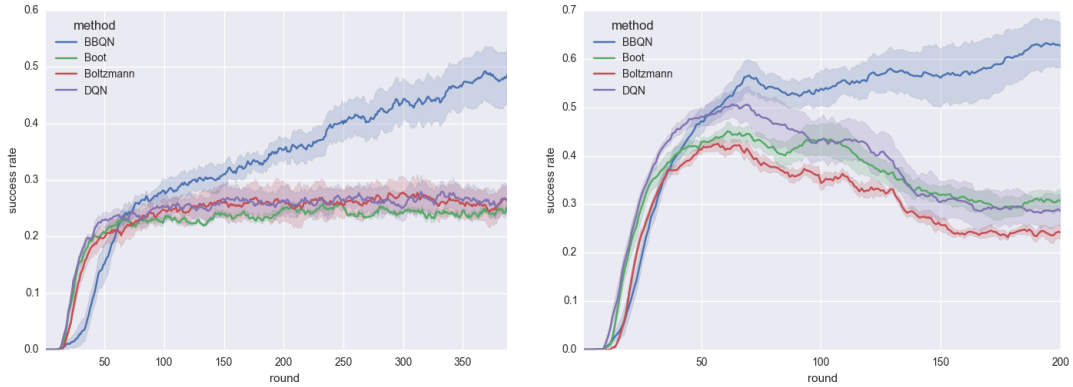
6.3 Architecture Details:

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

Each model has additional hyper-parameters. ϵ-greedy exploration requires an initial value of ϵ. Boltzmann exploration requires a temperature. The bootstrapping-based method of Osband et al. [2016] requires both a number of bootstrap heads and the 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.

6.4 Results:

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 models (Table 1). Note that the domain extension problem grows more difficult every 10 epochs, so sustained performance corresponds to getting better but declining performance doesn't imply that the policy grows worse. On both problems, no method achieves a single success absent replay buffer spiking. Interestingly,



(a) Full Domain (static)

(b) Domain Extension

Figure 2: Training plots with confidence intervals for the full domain and domain extension problems.

while RBS proved essential, both the BBQN and DQN both proved surprisingly insensitive to the precise number of dialogues used to spike the buffer.

	Final performance (Success Rate)				**Final performance (Reward)**			
Model	**BBQN**	**Bootstrap**	**Boltzmann**	**DQN**	**BBQN**	**Bootstrap**	**Boltzmann**	**DQN**
Full Domain	**.4877**	.2516	.2658	.2693	**9.984**	-0.130	0.418	0.866
Extension	**.6722**	.3170	.2435	.3503	**16.132**	-0.682	-3.464	4.756

Table 1: Final performance of trained models on 10k simulated dialogues, averaged over 5 runs

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 a purely greedy policy. Interestingly, non-zero values of ϵ resulted in higher regret and statistically indistinguishable final performance of the trained models. This might owe to the large state space of the dialogue task.

7 Related Work

Our paper touches several areas of research, namely Bayesian neural networks, reinforcement learning with deep Q-networks, Thompson Sampling, and dialogue systems. This work employs 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. Namely, Mnih et al. [2015] achieved super-human performance on Atari games, incorporating tricks such as experience replay [Lin, 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.

Efficient exploration remains one of the defining challenges in RL. 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], not much is known for the general case, especially when nonlinear 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 to reward the agent for discovering new dynamics of the environment. Osband et al. [2016] attempts to mine uncertainty information by training a neural network with multiple output *heads*. Each head is associated with a distinct subset of the data. This works for some

but not all Atari games, but does not improve over standard DQNs on our problem. Chapelle and Li [2011] present an empirical examination of Thompson sampling for contextual bandits. Thompson sampling has also proved helpful in RL with finite MDPs [Strens, 2000].

We build on the Bayes-by-backprop method due to Blundell et al. [2015], employing the reparameterization trick popularized by Kingma and Welling [2013], following a long history of variational treatments of neural networks [Hinton and Van Camp, 1993, Graves, 2011]. This paper also builds on prior work in task-oriented dialogue systems [Williams and Young, 2004, Gašić et al., 2010] and RL for learning dialogue policies [Levin et al., 1997, Singh et al., 2000, Gašić et al., 2010, Fatemi et al., 2016]. Our domain-extension experiments take inspiration from Gašic et al. [2014] and our agenda-based user simulator is modeled on Schatzmann et al. [2007].

8 Discussion

For learning dialog policies, BBQ networks explore with greater efficiency than traditional approaches. 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 BBQNs over other exploration strategies, we would like to extend this work to 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, Bellemare et al., 2016]. Conceivably, these approaches could be complementary. Our BBQN addresses uncertainty in the Q-value given the current policy, whereas curiosity addresses uncertainty over what else might be possible in policy space even if it would never be tried under the current policy. We anticipate a potential synergistic effect of combining the approaches.

References

John Asmuth, Lihong Li, Michael L. Littman, Ali Nouri, and David Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *UAI*, 2009.

Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *arXiv preprint arXiv:1606.01868*, 2016.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *ICML*, 2015.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *NIPS*, 2011.

Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with Gaussian processes. In *ICML*, 2005.

Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. Policy networks with two-stage training for dialogue systems. *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 *SIGDial*, 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. *Interspeech*, 2014.

Alex Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT*, 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:1605.09674*, 2016.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *JMLR*, 2010.

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

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

Esther Levin, Roberto Pieraccini, and Wieland Eckert. Learning dialogue strategies within the Markov decision process framework. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, 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*, 2011.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 1992.

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

Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

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

Jost Schatzmann, Blaise Thomson, and Steve Young. Statistical user simulation with a hidden agenda. *SIGDial*, 2007.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICML*, 2016. arXiv:1511.05952.

David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. Reinforcement learning for spoken dialogue systems. In *NIPS*, 2000.

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

Malcolm J. A. Strens. A Bayesian framework for reinforcement learning. In *ICML*, 2000.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2015.

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

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 1992.

Jason D Williams and Steve J Young. Characterizing task-oriented dialog using a simulated ASR channel. In *Interspeech*, 2004.