BBQ-Networks: Efficient Exploration in Deep Reinforcement Learning for Task-Oriented Dialogue Systems

Zachary Lipton[⋆] **Xiujun Li**[†] **Jianfeng Gao**[†] **Lihong Li**[‡∗] **Faisal Ahmed**[†] **Li Deng**[§∗]
[⋆]Carnegie Mellon University, Pittsburgh, PA, USA
[†]Microsoft Research, Redmond, WA, USA
[‡]Google Inc., Kirkland, WA, USA [§]Citadel, Seattle, WA, USA
[⋆]zlipton@cmu.edu, [†]{xiul,jfgao,fiahmed}@microsoft.com
[‡]lihongli.cs@gmail.com, [§]l.deng@ieee.org

Abstract

We present a new algorithm that significantly improves the efficiency of exploration for deep Q-learning agents in dialogue systems. Our agents explore via Thompson sampling, drawing Monte Carlo samples from a *Bayes-by-Backprop* neural network. Our algorithm learns much faster than common exploration strategies such as ϵ-greedy, Boltzmann, bootstrapping, and intrinsic-reward-based ones. Additionally, we show that spiking the replay buffer with experiences from just a few successful episodes can make Q-learning feasible when it might otherwise fail.

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. These bots typically carry out single-exchange conversations, but we aspire to develop more general dialogue agents, approaching the breadth of capabilities exhibited by human interlocutors. In this work, we consider task-oriented bots (Williams and Young 2004), agents charged with conducting a multi-turn dialogue to achieve some task-specific goal. In our case, we attempt to assist a user to book movie tickets.

For complex dialogue systems, it is often impossible to specify a good policy *a priori* and the dynamics of an environment may change over time. Thus, learning policies online and interactively via reinforcement learning (RL) has emerged as a popular approach (Singh et al. 2000; Gašić et al. 2010; Fatemi et al. 2016). Inspired by RL breakthroughs on Atari and board games (Mnih et al. 2015; Silver et al. 2016), we employ deep reinforcement learning (DRL) to learn policies for dialogue systems. Deep Q-network (DQN) agents typically explore via the ϵ-greedy heuristic, but when rewards are sparse and action spaces are large (as in dialogue systems), this strategy tends to fail. In our experiments, a randomly exploring Q-learner never experiences success in thousands of episodes.

We offer a new, efficient solution to improve the exploration of Q-learners. We propose a Bayesian exploration strategy that encourages a dialogue agent to explore state-action regions in which the agent is relatively uncertain in action selection. Our algorithm, the *Bayes-by-Backprop Q-network* (BBQN), explores via Thompson sampling, drawing Monte Carlo samples from a Bayesian neural network (Blundell et al. 2015). In order to produce the temporal difference targets for Q-learning, we must generate predictions from a frozen target network (Mnih et al. 2015). We show that using the maximum a posteriori (MAP) assignments to generate targets results in better performance (in addition to being computationally efficient). We also demonstrate the effectiveness of *replay buffer spiking* (RBS), a simple technique in which we pre-fill the experience replay buffer with a small set of transitions harvested from a naïve, but occasionally successful, rule-based agent. This technique proves essential for both BBQNs and standard DQNs.

We evaluate our dialogue agents on two variants of a movie-booking task. Our agent interacts with a user to book a movie. Success is determined at the end of the dialogue if a movie has been booked that satisfies the user. We benchmark our algorithm and baselines using an agenda-based user simulator similar to Schatzmann, Thomson, and Young (2007). To make the task plausibly challenging, our simulator introduces random mistakes to account for the effects of speech recognition and language understanding errors. In the first variant, our environment remains fixed for all rounds of training. In the second variant, we consider a non-stationary, domain-extension environment. In this setting, new attributes of films become available over time, increasing the diversity of dialogue actions available to both the user and the agent. Our experiments on both the stationary and domain-extension environments demonstrate that BBQNs outperform DQNs using either ϵ-greedy exploration, Boltzmann exploration, or the bootstrap approach introduced by Osband et al. (2016). Furthermore, the real user evaluation results consolidate the effectiveness of our approach that BBQNs are more effective than DQNs in exploration. Besides, we also show that all agents only work given replay buffer spiking, although the number of pre-filled dialogues can be small.

Task-Oriented dialogue systems

In this paper, we consider goal-oriented dialogue agents, specifically one that aims to help users to book movie tickets. Over the course of several exchanges, the agent gathers information such as movie name, theater and number of tick-

[∗]This work was done while LL & DL were with Microsoft.

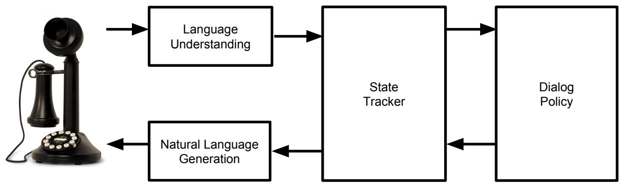


Figure 1: Components of a dialogue system

ets, and ultimately completes a booking. A typical dialogue pipeline is shown in Figure 1. In every turn of a conversation, the *language understanding* module converts raw text into structured semantic representations known as *dialog-acts*, which pass through the *state-tracker* to maintain a record of information accumulated from previous utterances. The *dialogue policy* then selects an action (to be defined later) which is transformed to a natural language form by a *generation* module. The conversation continues until the dialogue terminates. A numerical reward signal is used to measure the utility of the conversation. Details of this process are given below.

Dialog-acts Following Schatzmann, Thomson, and Young (2007), we represent utterances as dialog-acts, consisting of 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 semantic representation *request(ticket, moviename=Our Kind of Traitor, starttime=tonight, city=Seattle)*.

State tracker Other than information inferred from previous utterances, the state-tracker may also interact with a database, providing the policy with information such as how many movies match the current constraints. It then *delexicalizes* the dialog-act, allowing the dialogue policy to act upon more generic states. The tracked state of the dialogue, consisting of a representation of the conversation history and several database features, is passed on to the policy to select actions.

Actions Each action is a de-lexicalized *dialog-act*. In the movie-booking task, we consider a set of 39 actions. These include basic actions such as *greeting(), thanks(), deny(), confirm_question(), confirm_answer(), closing()*. Additionally, we add two actions for each slot: one to inform its value and the other to request it. The pipeline then flows back to the user. Any slots informed by the policy are then filled in by the state tracker. This yields a structured representation such as *inform(theater=Cinemark Lincoln Square)*, which is then mapped by a natural language generation module to a textual utterance, such as "This movie is playing tonight at Cinemark Lincoln Square."

The conversation process above can be naturally mapped to the reinforcement learning (RL) framework, as follows (Levin, Pieraccini, and Eckert 1997). The RL agent navigates a Markov decision process (MDP), interacting with its environment over a sequence of discrete steps (Sutton and Barto 1998). At step $t \in \{1, 2, \ldots\}$, the agent observes the current state s_t, and chooses some action a_t 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. In this work, we assume that the set of actions, denoted $\mathcal{A}$, is finite. In our dialogue scenario, the state-tracker produces states, actions are the de-lexicalized dialog-acts described earlier, state transitions are governed by the dynamics of the conversation, and a properly defined reward function is used to measure the degree of success of a dialogue. In our experiment, for example, success corresponds to a reward of 40, failure to a reward of -10, and we apply a per-turn penalty of -1 to encourage pithy exchanges.

The goal of RL is to find an optimal policy to maximize long-term reward. The Q-function measures, for every state-action pair (s, a), the maximum expected cumulative discounted reward achieved by choosing a in s and then following an optimal policy thereafter: $Q^*(s, a) = \max_\pi \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s, a_t = a\right]$, where $\gamma \in (0, 1)$ is a discount factor. Owing to large state spaces, most practical reinforcement learners approximate the Q-function by some parameterized model $Q(s, a; \theta)$. An example, as we used in this paper, is a neural network, where θ represents the set of weights to be learned. Once a good estimate of θ is found so that $Q(\cdot, \cdot; \theta)$ is a good approximation of $Q(\cdot, \cdot)$, the greedy policy, $\pi(s; \theta) = \arg\max_a Q(s, a; \theta)$, is a near-optimal policy (Sutton and Barto 1998). A popular way to learn a neural-network-based Q-function is known as DQN (Mnih et al. 2015); see the appendix for more details.

Bayes-by-Backprop

Bayes-by-Backprop (Blundell et al. 2015) captures uncertainty information from neural networks by maintaining a probability distribution over the weights in the network. For simplicity, we explain the idea for multilayer perceptrons (MLPs). An L-layer MLP for model $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w})$ is parameterized by weights $\boldsymbol{w} = \{W_l, b_l\}_{l=1}^{L}$: $\hat{\boldsymbol{y}} = W_L \cdot \phi(W_{L-1} \cdot \ldots \cdot \phi(W_1 \cdot \boldsymbol{x} + b_1) + \ldots + b_{L-1}) + b_L$, where ϕ is an activation function such as sigmoid, tanh, or rectified linear unit (ReLU). In standard neural network training, weights are optimized by SGD to minimize a loss function such as squared error.

With Bayes-by-Backprop, we impose a prior distribution over the weights, $p(\boldsymbol{w})$, and learn the full posterior distribution, $p(\boldsymbol{w}|\mathcal{D}) \propto p(\boldsymbol{w})p(\mathcal{D}|\boldsymbol{w})$, given training data $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$. In practice, however, computing an arbitrary posterior distribution can be intractable. So, we instead approximate the posterior by a variational distribution, $q(\boldsymbol{w}|\theta)$. In this work, we choose q to be a Gaussian with diagonal covariance, i.e., each weight w_i is sampled from $\mathcal{N}(\mu_i, \sigma_i^2)$. 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 a D-dimensional weight vector $\boldsymbol{w}$.

We learn these parameters by minimizing *variational free energy* (Hinton and Van Camp 1993), the KL-divergence between the variational approximation $q(\boldsymbol{w}|\theta)$ and the posterior $p(\boldsymbol{w}|\mathcal{D})$:

$$\begin{aligned}\theta^* &= \text{argmin}_\theta \text{KL}[q(\boldsymbol{w}|\theta)||p(\boldsymbol{w}|\mathcal{D})] \\ &= \text{argmin}_\theta \Big\{ \text{KL}[q(\boldsymbol{w}|\theta)||p(\boldsymbol{w})] - \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\log p(\mathcal{D}|\boldsymbol{w})] \Big\} .\end{aligned}$$

When $\boldsymbol{w}$ is sampled from q, the above objective function can be estimated by its empirical version: $f(\mathcal{D}, \theta) = \log q(\boldsymbol{w}|\theta) - \log p(\boldsymbol{w}) - \log p(\mathcal{D}|\boldsymbol{w})$. It can be minimized by SGVB, using the reparametrization trick popularized by Kingma and Welling (2013). See Appendix Bayes-by-Backprop for more details.

BBQ-networks

We are now ready to introduce *BBQN*, our algorithm for learning dialogue policies with deep learning models. BBQN builds upon the deep Q-network, or DQN (Mnih et al. 2015), and uses a Bayesian neural network to approximate the Q-function and the uncertainty in its approximation. Since we work with fixed-length representations of dialogues, we use an MLP, but extending our methodology to recurrent or convolutional neural networks is straightforward.

Action selection A distinct feature of BBQN is that it explicitly quantifies uncertainty in the Q-function estimate, which can be used to guide exploration. In DQN, the Q-function is represented by a network with parameter $\boldsymbol{w}$. BBQN, in contrast, maintains a distribution q over $\boldsymbol{w}$. As described in the previous section, q is a multivariante Gaussian with diagonal covariance, parameterized by $\theta = \{(\mu_i, \rho_i)\}_{i=1}^D$. In other words, a weight w_i has a posterior distribution q that is $\mathcal{N}(\mu_i, \sigma_i^2)$ where $\sigma_i = \log(1 + \exp(\rho_i))$.

Given a posterior distribution q over $\boldsymbol{w}$, a natural and effective approach to exploration is posterior sampling, or Thompson Sampling (Thompson 1933; Chapelle and Li 2011; Osband, Russo, and Roy 2013), in which actions are sampled according to the posterior probability that they are optimal in the current state. Formally, given a state s_t and network parameter θ_t in step t, an action a is selected to be a_t with the probability $\Pr(a_t = a|s_t, \theta_t) =$

$$\int_{\boldsymbol{w}} \mathbf{1}\{\, Q(s_t, a; \boldsymbol{w}) > Q(s, a'; \boldsymbol{w}), \forall a' \neq a\} \cdot dq(\boldsymbol{w}|\theta_t) \,. \quad (1)$$

Computing these probabilities is usually difficult, but fortunately all we need is a *sample* of an action from the corresponding multinomial distribution. To do so, we first draw $\boldsymbol{w}_t \sim q(\cdot|\theta_t)$, then set $a_t = \arg\max_a Q(s_t, a; \boldsymbol{w}_t)$. It can be verified that this process samples actions with the same probabilities given in the Equation 1. We have also considered integrating the ϵ-greedy approach, exploring by Thompson sampling with probability $1-\epsilon$ and uniformly at random with probability ϵ. But empirically, uniform random exploration confers no supplementary benefit for our task.

BBQN The BBQN is initialized by a prior distribution p over $\boldsymbol{w}$. It consists of an isotropic Gaussian whose variance σ_p^2 is a single hyper-parameter introduced by our model. We initialize the variational parameters to match the prior. So $\boldsymbol{\mu}$ is initialized to the zero vector $\mathbf{0}$ and the variational standard deviation $\boldsymbol{\sigma}$ matches the prior σ_p for each weight. Note that unlike conventional neural networks, we need not assign the weights randomly because 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 experience buffer fills, the mean squared error starts to dominate the objective function and the variational distribution moves further from the prior.

Given experiences of the form $\mathcal{T} = \{(s, a, r, s')\}$ consisting of transitions collected so far, we apply a Q-learning approach to optimize the network parameter, in a way similar to DQN (Mnih et al. 2015). To do so, we maintain a frozen, but periodically updated, copy of the same BBQN, whose parameter is denoted by $\tilde{\theta} = \{(\tilde{\mu}_i, \tilde{\rho}_i)\}_{i=1}^D$. For any transition $(s, a, r, s') \in \mathcal{T}$, this network is used to compute a target value y for $Q(s, a; \theta)$, resulting in a regression data set $\mathcal{D} = \{(x, y)\}$, for $x = (s, a)$. We then apply the Bayes-by-backprop method described in the previous section to optimize θ, until it converges when $\tilde{\theta}$ is replaced by θ. There are two ways to generate the target value y.

The first uses a Monte Carlo sample from the frozen network, $\tilde{\boldsymbol{w}} \sim q(\cdot|\tilde{\theta})$, to compute the target y: $y = r + \gamma \max_{a'} Q(s', a'; \tilde{\boldsymbol{w}})$. To speed up training, for each mini-batch, we draw one sample of $\tilde{\boldsymbol{w}}$ for target generation, and one sample of $\boldsymbol{w}$ for sample-based variational inference (see previous section). With this implementation, the training speeds of BBQN and DQN are roughly equivalent.

The second uses maximum a posterior (MAP) estimate to compute y: $y = r + \gamma \max_{a'} Q(s', a'; \tilde{\mu})$. This computationally more efficient choice is motivated by the observation that, since we only require the uncertainty estimates for exploration, it may not be necessary to sample from the frozen network for synthesizing targets. Furthermore, early in training, the predictive distribution of the networks has high variance, resulting in a large amount of noise in target values that can slow down training.

BBQN with intrinsic reward Variational Information Maximizing Exploration (VIME) (Houthooft et al. 2016a) introduces an exploration strategy based on maximizing the information gain about the agent's belief of environment dynamics. It adds an *intrinsic reward* bonus to the reward function, which quantifies the agent's *surprise*: $r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta \mathbb{D}_{KL}[p(\theta|\xi_t, a_t, s_{t+1})||p(\theta|\xi_t)]$, (where ξ_t is defined as the history of the agent up until time step t: $\xi_t = \{s_1, a_1, ..., s_t\}$), and has demonstrated strong empirical performance. We explore a version of BBQNs that incorporates the intrinsic reward from VIME, terming the approach BBQN-VIME-MC/MAP. The BBQN-VIME variations encourage the agents to explore the state-action regions that are relatively unexplored and in which BBQN is relatively uncertain in action selection. In our full-domain experiment, both BBQN and BBQN-VIME variations achieve similar performance with no significant difference, but in domain-extension experiments, we observe that BBQN-VIME-MC slightly outperforms BBQN-MAP.

Replay buffer spiking In reinforcement learning, there are multiple sources of uncertainty. These include uncertainty over the parameters of our model and uncertainty over unseen parts of the environment. BBQN addresses parameter uncertainty but it can struggle given extreme reward sparsity. Researchers use various techniques to accelerate learning in these settings. One approach is to leverage prior knowledge, as by reward shaping or imitation learning. Our approach falls into this category. Fortunately, in our setting, it's easy to produce a few successful dialogues manually. Even though the manual dialogues do not follow an optimal policy, they contain some successful movie bookings, so they indicate the existence of the large ($+40$) reward signal. Pre-filling the replay buffer with these experiences dramatically improves performance (Figure 3). For these experiments, we construct a simple rule-based agent that, while sub-optimal (18.3% success rate), achieves success sometimes. In each experiment, we harvest 100 dialogues of experiences from the rule-based agent, adding them to the replay buffer. We find that, in on our task, RBS is essential for both BBQN and DQN approaches. Interestingly, performance does not strictly improve with the number pre-filled dialogues (Figure 3). Note that replay buffer spiking is different from imitation learning. RBS works well with even a small number of warm-start dialogues, suggesting that it is helpful to communicate even the very *existence* of a big reward. We find that even one example of a successful dialogue in the replay buffer could successfully jump-start a Q-learner.

Experiments

We evaluate our methods on two variants of the movie-booking task. In our experiments, we adapt the publicly available[1] simulator described in Li et al. (2016). In the first, the agent interacts with the user simulator over 400 rounds. Each round consists of 50 simulated dialogues, followed by 2 epochs of training. All slots are available starting from the very first episode. In the second, we test each model's ability to adapt to domain extension by 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 order. For each added slot, the state space and action space grow accordingly. This experiment terminates after 200 rounds. In both experiments, quantifying uncertainty in the network weights is important to guide effective exploration.

To represent the state of the dialogue 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; and (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 domain-extension experiments, features corresponding to unseen slots take value 0 until they are seen. When domain is extended, we add features and corresponding weights to input layer, initializing the new weights to 0 (or $\mu_i = 0$, $\sigma_i = \sigma_{prior}$ for BBQN), a trick due to Lipton, Vikram, and McAuley (2015).

[1] `https://github.com/MiuLab/UserSimulator`

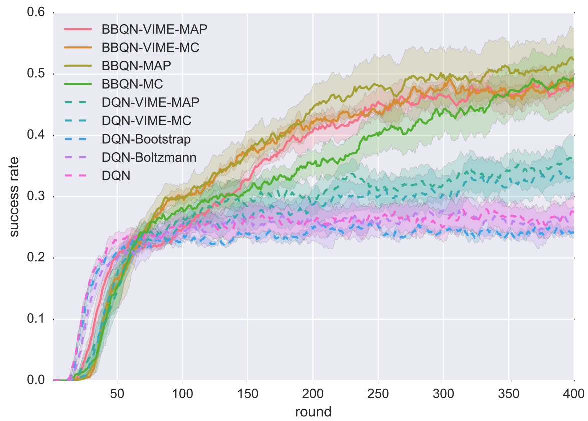
Agents	Full Domain		Domain Extension	
	Success Rate	Reward	Success Rate	Reward
BBQN-VIME-MAP	0.4856	9.8623	0.6813	15.8223
BBQN-VIME-MC	0.4941	10.4268	**0.7120**	**17.6261**
BBQN-MAP	**0.5031**	**10.7093**	0.6852	17.3230
BBQN-MC	0.4877	9.9840	0.6722	16.1320
DQN-VIME-MAP	0.3893	5.8616	0.3751	4.9223
DQN-VIME-MC	0.3700	4.9990	0.3675	4.8270
DQN-Bootstrap	0.2516	-0.1300	0.3170	-0.6820
DQN-Boltzmann	0.2658	0.4180	0.2435	-3.4640
DQN	0.2693	0.8660	0.3503	4.7560

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

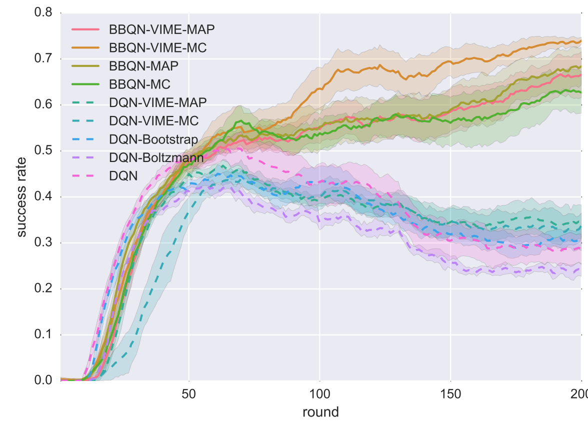
Training details For training, we first use a naive but occasionally successful rule-based agent for RBS. All experiments use 100 dialogues to spike the replay buffer. We note that experiments showed models to be insensitive to the precise number. After each round of 50 simulated dialogues, the agent 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. There are two reasons for proceeding in 50-dialog spurts, rather than updating one mini-batch per turn. First, in a deployed system, real-time updates might not be realistic. Second, we train for more batches per new turn than is customary in DQN literatures owing to the economic considerations: computational costs are negligible, while failed dialogues either consume human labor (in testing) or confer opportunity costs (in the wild).

Baseline methods To demonstrate the efficacy of BBQN, we compare against ϵ-greedy in 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 is parameterized by the Boltzmann *temperature*. We also compare to the bootstrapping 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 0.5. We evaluate all four methods on both the full domain (static) learning problem and on the domain extension problem.

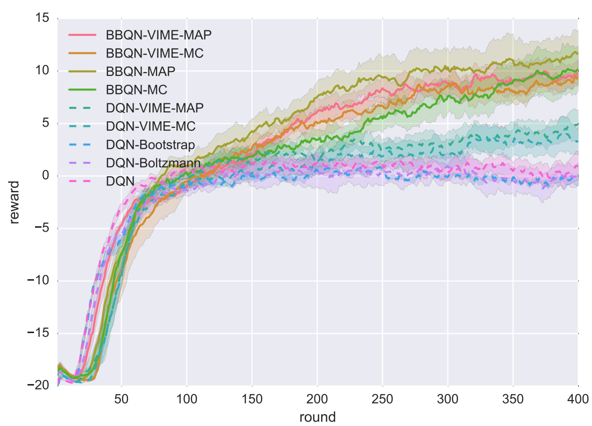
We also tried comparing against Gaussian processes (GP) based approaches. However, in our setting, due to the high-dimensional inputs and large number of time steps, we were unable to get good results. In our experiments, the computation and memory requirement grow quadratically over



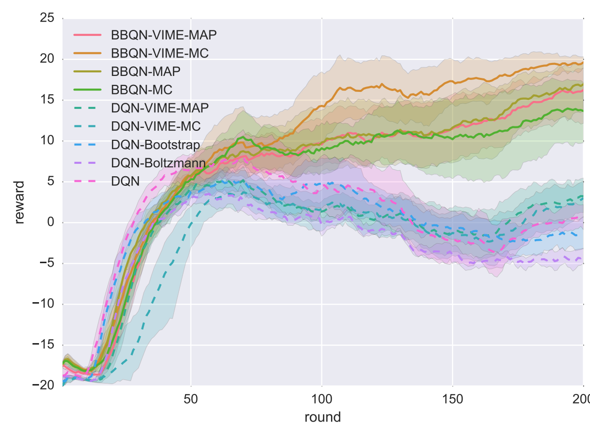
(a) Full domain (success rate)



(b) Domain extension (success rate)



(c) Full domain (reward)



(d) Domain extension (reward)

Figure 2: Training plots with confidence intervals for the full domain (all slots available from start) and domain extension problems (slots added every 10 rounds).

time, and memory starts to explode at the 10th (simulation) round. Limiting data size for GP was not helpful. Furthermore, in contrast to Gašić et al. (2010) where the state is 3-dimensional, our experiments have 268-dimensional states, making scalability an even bigger challenge. A recent paper (Fatemi et al. 2016) compares deep RL (both policy gradient and Q-learning) to GP-SARSA (Engel, Mannor, and Meir 2005) on a simpler dialogue policy learning problem. In order to make Gaussian processes computationally tractable, they rely on sparsification methods (Engel, Mannor, and Meir 2005), gaining computation efficiency at the expense of accuracy. Despite this undertaking to make GPs feasible and competitive, they found that deep RL approaches outperform GP-SARSA with respect to final performance, regret, and computational expense (by wall-clock). While we consider Gaussian processes to be an evolving area, it is worthwhile to try the Gaussian processes with sparsification methods to compare with deep RL approaches as future work.

Architecture details All models are MLPs with ReLU activations. Each network has 2 hidden layers with 256 hidden nodes each. We optimize over parameters using Adam (Kingma and Ba 2015) with a batch size of 32 and initial learning rate of 0.001, determined by a grid search. To avoid biasing the experiments towards our methods, we determine common hyper-parameters using standard DQN. Because BBQN confers regularization, we equip DQN models with dropout regularization of 0.5, shown by Blundell et al. (2015) to confer comparable predictive performance on holdout data.

Each model has additional hyper-parameters. For example, ϵ-greedy exploration requires an initial value of ϵ and an attenuation schedule. 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 BBQN requires that we determine the variance of the Gaussian prior distribution and the variance of the Gaussian error distribution.

Simulation results As shown in Figure 2, BBQN variants perform better than the baselines. In particular, BBQN-MAP performs the best on the full domain setting, BBQN-VIME-MC achieves the best performance on the domain extension setting, with respect to cumulative successes during training and final performance of the trained models (Table 1). Note that the domain extension problem becomes more difficult every 10 epochs, so sustained performance corresponds to getting better, while declining performance does not imply the policy becomes worse. On both problems, no method achieves a single success absent RBS. Evaluating our best algorithm (BBQN-MAP) using 0, 100, and 1000 RBS dialogues (Figure 3), we find that using 1000 (as compared to 100) dialogues, our agents learn quickly but that their long-term performance is worse. One heuristic to try in the future may be to discard pre-filled experiences after meeting some performance threshold.

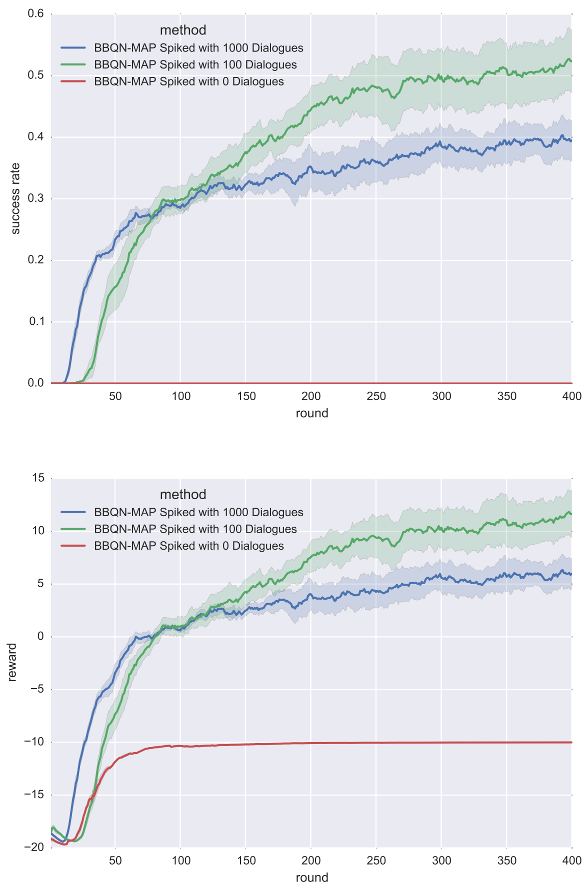


Figure 3: RBS with 100 dialogues improves both success rate (top) and reward (bottom).

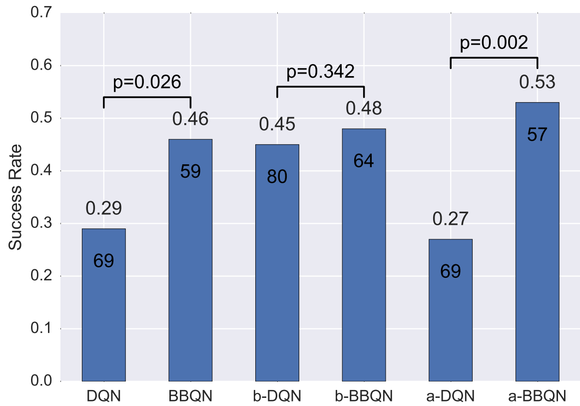
We also considered that perhaps some promising trajectories might never be sampled by the BBQN. Thus, we constructed an experiment exploring via a hybridization of the BBQN's Thompson sampling with the ϵ-greedy approach. With probability $1-\epsilon$, the agent selects an action by Thompson sampling given one Monte Carlo sample from the BBQN and with probability ϵ the agent selects an action uniformly at random. However, the uniformly random exploration confers no additional benefit.

Human evaluation We evaluate the agents trained using simulated users against real users, recruited from the authors' affiliation. We conducted the study using the DQN and BBQN-MAP agents. In the full-domain setting, the agents were trained with all the slots. In the domain-extension setting, we first picked DQN (b-DQN) and BBQN (b-BBQN) agents before the domain extension at training epoch 40 and the performance of these two agents is tied, nearly 45% success rate. From training epoch 40, we started to introduce new slots, and we selected another two agents (a-DQN and a-BBQN) at training epoch 200. In total, we compare three agent pairs: {DQN, BBQN} for full domain, {b-DQN, b-BBQN} from before domain extension, and {a-DQN, a-BBQN} from after domain extension. In the real user study, for each dialogue session, we select one of six agents randomly to converse with a user. We present the user with a user goal sampled from our corpus. At the end of each dialogue session, the user was asked to give a rating on a scale from 1 to 5 based on the naturalness, coherence, and task-completion capability of the agent (1 is the worst rating, 5 is the best). In total, we collected 398 dialogue sessions. Figure 4a presents the performance of these agents against real users in terms of success rate. Figure 4b shows the comparison in user ratings. In the full-domain setting, the BBQN agent is significantly better than the DQN agent in terms of success rate and user rating. In the domain-extension setting, before domain extension, the performance of both agents (b-DQN and b-BBQN) is tied; after domain extension, the BBQN (a-BBQN) agent significantly outperforms the DQN (a-DQN) in terms of success rate and user rating.

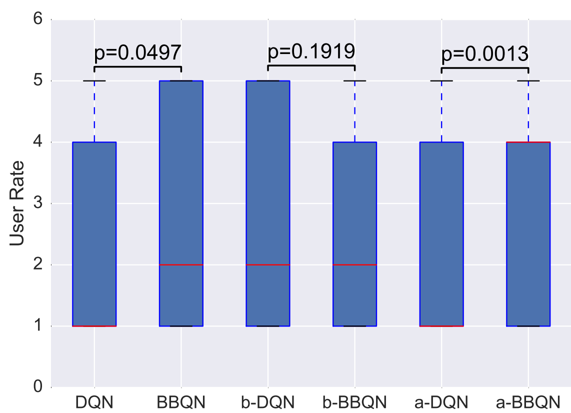
Related work

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). Recently, Mnih et al. (2015) achieved super-human performance on Atari games using deep Q-learning and incorporating techniques such as experience replay (Lin 1992).

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, Ortner, and Auer 2010; Li et al. 2011; Osband, Russo, and Roy 2013), less is known for the general case, especially when general nonlinear function approximation is used. The first DQN papers relied upon the ϵ-greedy exploration heuristic (Mnih et al. 2015). More recently, Stadie, Levine, and Abbeel (2015) and Houthooft et al. (2016a; 2016b) introduced approaches to encourage exploration by perturbing the reward function. Osband et



(a) Distribution of Success Rate



(b) Distribution of User Ratings

Figure 4: Performance of BBQN agent versus DQN agent tested with real users, number of tested dialogues and p-values are indicated on each bar (difference in mean is significant with $p < 0.05$).

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 Atari games, but does not confer a benefit for us. Chapelle and Li (2011) empirically examine Thompson sampling, one of the oldest exploration heuristics (Thompson 1933), for contextual bandits, which is later shown to be effective for solving finite-state MDPs (Strens 2000; Osband, Russo, and Roy 2013).

We build on the Bayes-by-backprop method of Blundell et al. (2015), employing the reparameterization trick popularized by Kingma and Welling (2013), and following a long history of variational treatments of neural networks (Hinton and Van Camp 1993; Graves 2011). After we completed this work, Kirkpatrick et al. (2017) independently investigated parameter uncertainty for deep Q-networks to mitigate catastrophic forgetting issues. Blundell et al. (2015) consider Thompson sampling for contextual bandits, but do not consider the more challenging case of MDPs. Our paper also builds on prior work in task-oriented dialogue systems (Williams and Young 2004; Gašić et al. 2010; Wen et al. 2016) and RL for learning dialogue policies (Levin, Pieraccini, and Eckert 2000; Singh et al. 2000; Williams and Young 2007; Gašić et al. 2010; Fatemi et al. 2016). Our domain-extension experiments take inspiration from Gašic et al. (2014) and our user simulator is modeled on Schatzmann, Thomson, and Young (2007).

Conclusions

For learning dialogue policies, BBQNs explore with greater efficiency than traditional approaches. The results are similarly strong for both static and domain extension experiments in simulation and real human evaluation. Additionally, we showed that we can benefit from combining BBQ-learning with other, orthogonal approaches to exploration, such as those 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, or previously rarely seen states (Stadie, Levine, and Abbeel 2015; Houthooft et al. 2016a; Houthooft et al. 2016b; Bellemare et al. 2016). Our BBQN addresses uncertainty in the Q-value given the current policy, whereas curiosity addresses uncertainty of the dynamics of under-explored parts of the environment. Thus there is a synergistic effect of combining the approaches. On the domain extension task, BBQN-VIME proved especially promising, outperforming all other methods. We see several promising paths for future work. Notably, given the substantial improvements of BBQNs over other exploration strategies, we would like to extend this work to popular deep reinforcement learning benchmark tasks (Atari, etc.) and other domains, like robotics, where the cost of exploration is high, to see if it confers a comparably dramatic improvement.

References

[Asmuth et al. 2009] Asmuth, J.; Li, L.; Littman, M. L.; Nouri, A.; and Wingate, D. 2009. A Bayesian sampling approach to exploration in reinforcement learning. In *UAI*.

[Bellemare et al. 2016] Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In *NIPS*.

[Blundell et al. 2015] Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; and Wierstra, D. 2015. Weight uncertainty in neural networks. In *ICML*.

[Chapelle and Li 2011] Chapelle, O., and Li, L. 2011. An empirical evaluation of Thompson sampling. In *NIPS*.

[Engel, Mannor, and Meir 2005] Engel, Y.; Mannor, S.; and Meir, R. 2005. Reinforcement learning with Gaussian processes. In *ICML*.

[Fatemi et al. 2016] Fatemi, M.; Asri, L. E.; Schulz, H.; He, J.; and Suleman, K. 2016. Policy networks with two-stage training for dialogue systems. *arXiv:1606.03152*.

[Gašić et al. 2010] Gašić, M.; Jurčíček, F.; Keizer, S.; Mairesse, F.; Thomson, B.; Yu, K.; and Young, S. 2010.

Gaussian processes for fast policy optimisation of pomdp-based dialogue managers. In *SIGDial*.

[Gašic et al. 2014] Gašic, M.; Kim, D.; Tsiakoulis, P.; Breslin, C.; Henderson, M.; Szummer, M.; Thomson, B.; and Young, S. 2014. Incremental on-line adaptation of pomdp-based dialogue managers to extended domains. *Interspeech*.

[Graves 2011] Graves, A. 2011. Practical variational inference for neural networks. In *NIPS*.

[Hinton and Van Camp 1993] Hinton, G. E., and Van Camp, D. 1993. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT*.

[Houthooft et al. 2016a] Houthooft, R.; Chen, X.; Chen, X.; Duan, Y.; Schulman, J.; Turck, F. D.; and Abbeel, P. 2016a. VIME: Variational information maximizing exploration. In *NIPS*.

[Houthooft et al. 2016b] Houthooft, R.; Chen, X.; Duan, Y.; Schulman, J.; De Turck, F.; and Abbeel, P. 2016b. Curiosity-driven exploration in deep reinforcement learning via Bayesian neural networks. *arXiv:1605.09674*.

[Jaksch, Ortner, and Auer 2010] Jaksch, T.; Ortner, R.; and Auer, P. 2010. Near-optimal regret bounds for reinforcement learning. *JMLR*.

[Kakade 2003] Kakade, S. 2003. *On the Sample Complexity of Reinforcement Learning*. Ph.D. Dissertation, Gatsby Computational Neuroscience Unit, UCL, UK.

[Kingma and Ba 2015] Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR*.

[Kingma and Welling 2013] Kingma, D. P., and Welling, M. 2013. Auto-encoding variational Bayes. *arXiv:1312.6114*.

[Kirkpatrick et al. 2017] Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*.

[Levin, Pieraccini, and Eckert 1997] Levin, E.; Pieraccini, R.; and Eckert, W. 1997. Learning dialogue strategies within the Markov decision process framework. In *IEEE Workshop on Automatic Speech Recognition and Understanding*.

[Levin, Pieraccini, and Eckert 2000] Levin, E.; Pieraccini, R.; and 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.

[Li et al. 2011] Li, L.; Littman, M. L.; Walsh, T. J.; and Strehl, A. L. 2011. Knows what it knows: A framework for self-aware learning. *Machine Learning*.

[Li et al. 2016] Li, X.; Lipton, Z. C.; Dhingra, B.; Li, L.; Gao, J.; and Chen, Y.-N. 2016. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*.

[Lin 1992] Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*.

[Lipton, Vikram, and McAuley 2015] Lipton, Z. C.; Vikram, S.; and McAuley, J. 2015. Capturing meaning in product reviews with character-level generative text models. *arXiv:1511.03683*.

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*.

[Osband et al. 2016] Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep exploration via bootstrapped DQN. In *NIPS*.

[Osband, Russo, and Roy 2013] Osband, I.; Russo, D.; and Roy, B. V. 2013. (More) efficient reinforcement learning via posterior sampling. In *NIPS*.

[Schatzmann, Thomson, and Young 2007] Schatzmann, J.; Thomson, B.; and Young, S. 2007. Statistical user simulation with a hidden agenda. *SIGDial*.

[Schaul et al. 2016] Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized experience replay. In *ICLR*. arXiv:1511.05952.

[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*.

[Singh et al. 2000] Singh, S. P.; Kearns, M. J.; Litman, D. J.; and Walker, M. A. 2000. Reinforcement learning for spoken dialogue systems. In *NIPS*.

[Stadie, Levine, and Abbeel 2015] Stadie, B. C.; Levine, S.; and Abbeel, P. 2015. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv:1507.00814*.

[Strens 2000] Strens, M. J. A. 2000. A Bayesian framework for reinforcement learning. In *ICML*.

[Sutton and Barto 1998] Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT Press.

[Thompson 1933] Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*.

[Van Hasselt, Guez, and Silver 2015] Van Hasselt, H.; Guez, A.; and Silver, D. 2015. Deep reinforcement learning with double Q-learning. In *AAAI*.

[Watkins and Dayan 1992] Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine Learning*.

[Wen et al. 2016] Wen, T.-H.; Gasic, M.; Mrksic, N.; Rojas-Barahona, L. M.; Su, P.-H.; Ultes, S.; Vandyke, D.; and Young, S. 2016. A network-based end-to-end trainable task-oriented dialogue system. *arXiv:1604.04562*.

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

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

Appendices

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 $\pi : \mathcal{S} \to \mathcal{A}$ 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). Owing to large state spaces, 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.

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 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, 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.

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

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

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

$$\mathcal{L}(\theta_t) = \mathbb{E}\left[(y_t - Q(s_t, a_t; \theta_t))^2\right] \qquad (2)$$

for $y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_t)$. Traditionally, the Q-function is trained by stochastic approximation, estimating the loss on each experience as it is encountered, yielding the update:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(y_t - Q(s_t, a_t; \theta_t))\nabla Q(s_t, a_t; \theta_t). \qquad (3)$$

A few tricks improve the effectiveness of DQNs. First, *experience replay* maintains a buffer of experiences, training off-policy on randomly selected mini-batches (Lin 1992; Mnih et al. 2015). Second, it's common to periodically cache DQN parameters parameters, using the stale parameters to compute the training targets y_t.

Other techniques such as double deep Q-learning (Van Hasselt, Guez, and Silver 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. In this work, we improve upon greedy exploration strategies by using uncertainty information (in the predicted Q values) to make more intelligent exploration choices.

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:

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

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 $\boldsymbol{w}^{\boldsymbol{MAP}} = \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{D})$. This yields ℓ_2^2 regularization for Gaussian prior or ℓ_1 regularization for Laplace prior:

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

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. 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. 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 $p(\boldsymbol{w}|\mathcal{D})$

$$\begin{aligned}\theta^* &= \text{argmin}_\theta \text{KL}[q(\boldsymbol{w}|\theta)||p(\boldsymbol{w}|\mathcal{D})] \\ &= \text{argmin}_\theta \int q(\boldsymbol{w}|\theta) \log \frac{q(\boldsymbol{w}|\theta)}{p(\boldsymbol{w})p(\mathcal{D}|\boldsymbol{w})} \text{d}\boldsymbol{w} \\ &= \text{argmin}_\theta \text{KL}[q(\boldsymbol{w}|\theta)||p(w)] - \mathbb{E}_{q(\boldsymbol{w}|\theta)}[\log p(\mathcal{D}|\boldsymbol{w})].\end{aligned} \tag{5}$$

The expression minimized is termed by (Hinton and Van Camp 1993) the *variational free energy*

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

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 $q(\boldsymbol{w}|\theta)$. Therefore, Hinton and Van Camp (1993) describe this term as a penalty on the description length of weights. Assuming a Gaussian predictive distribution, the rightmost term is simply the expected square loss. Sampling from q, our cost function is $f(\mathcal{D}, \theta) = \log q(\boldsymbol{w}|\theta) - \log p(\boldsymbol{w}) - \log p(\mathcal{D}|\boldsymbol{w})$. 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.

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.

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 $\mathcal{N}(0, I)^D$.

Sample Dialogues

Below shows one success dialogue example generated by the rule-based agent and BBQN agent interacting with user simulator in the movie-booking domain.

User Goal

```
{
  "inform_slots": {                    "request_slots": {
     "city": "Seattle"                   "ticket": "?"
     "numberofpeople": "2"             }
     "theater": "regal meridian 16"
     "date": "tomorrow"
     "moviename": "Zoolander 2"
  }
}
```

Rule-based agent

User: Can I get some tickets for Zoolander 2?
Agent: What movie are you interested in?
User: I want to watch Zoolander 2.
Agent: What time would you like to see it?
User: I want to watch at 9:25 pm.
Agent: What city you would like?
User: I want to watch at Seattle.
Agent: What date would you like to watch it?
User: I want to set it up tomorrow.
Agent: Which theater would you like?
User: I want to watch at regal meridian 16.
Agent: How many tickets do you need?
User: I want 2 tickets please!
Agent: Okay, I was able to order 2 tickets for you to see Zoolander 2 tomorrow at regal meridian 16 theater in Seattle at 9:25 pm.
User: Thank you.
Agent: Thank you.

BBQN agent

User: Can I get tickets for Zoolander 2 tomorrow?
Agent: Which city would you like?
User: I want to watch it at Seattle.
Agent: Which theater do you like?
User: Regal Meridian 16.
Agent: How many tickets do you purchase?
User: I wanna 2 tickets please!
Agent: What time do you like?
User: 9:25 pm.
Agent: Okay, I was able to order 2 tickets for you to see Zoolander 2 tomorrow at regal meridian 16 theater in Seattle at 9:25 pm.
User: Thank you.
Agent: Thank you.