Bayes By Backprop Neural Networks for Dialogue Management

Christopher Tegho

Queens' College

MPhil Machine Learning,
Speech and Language Technology

August 11th, 2017
Cambridge University

I, Christopher Tegho of Queens' college, being a candidate for the M.Phil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Number of words: 11997

Signed

Chris Tegho

Date

August 11th, 2017

Abstract

In dialogue management for statistical spoken dialogue systems, an agent learns a policy that maps a belief state to an action for the system to perform. Efficient exploration is key to successful dialogue policy estimation. Current deep reinforcement learning methods are very promising but rely on ε-greedy exploration, which is not as sample efficient as methods that use uncertainty estimates, such as Gaussian Process SARSA (GPSARSA). This thesis implements Bayes-By-Backpropagation, a method to extract uncertainty estimates from deep Q-networks (DQN). These uncertainty estimates are used to guide exploration. We show that Bayes-By-Backpropagation DQN (BBQN) achieves more efficient exploration and faster convergence to an optimal policy than ε-greedy based methods, and reaches performance comparable to the state of the art in policy optimization, namely GPSARSA, especially when evaluated on more complex domains, and without the high computational complexity of Gaussian Processes. We also implement α-divergences, variational dropout, and minimizing the negative log likelihood as other means to extract uncertainty estimates from DQN, and compare performance to BBQN and DQN. This work is carried within in the Cambridge University Engineering Department dialogue systems toolkit, CUED-pydial.

Contents

Acknowledgments

I would like to thank my supervisor Dr Milica Gašić whose guidance and support were very valuable for completing this research. I would also like to thank Paweł Budzianowski for his guidance, presence, ideas and thoughts about my work. I thank the Fonds de recherche du Québec – Nature et technologies (FRQNT) for partially funding my MPhil.

Introduction

0.1 Motivation

Spoken Dialogue Systems (SDS) allow human users to interact with computers through speech. SDS have became a common deployment in the speech interfaces in mobile phones, and are gaining greater commercial use. Building robust SDS is challenging, and traditional approaches have relied on manually implementing hand-coded rules to guide the system's response. For real-world dialogue systems, the number of such rules is very large, and implementing them is not cost-effective. A statistical approach to SDS on the other hand automates learning of dialogue rules, and increases robustness to errors that arise from imperfect speech recognition and language understanding [12].

The partially observable Markov decision process (POMDP) is used to model dialogue and to automatically learn dialogue rules. The POMDP framework overcomes noise injected in the state of the system by assuming the state is only partially observable and depends on a noisy observation. A distribution over states is maintained, which is called the belief state. The system takes as input the belief state, and a dialogue policy maps this input into an appropriate action at every dialogue turn.

Reinforcement leaning (RL) is used to learn a policy that maximizes the expected sum of rewards received after visiting a state. An action-state value function Q is computed for this purpose. However, the state-action space for dialogue systems is large, and a Q-function approximation is necessary to achieve efficient exploration, which is key to successful dialogue policy optimization. To explore the environment, an ε-greedy policy can be employed, where a greedy action is taken with respect to the estimated Q function with probability $1 - \varepsilon$, and a random one with probability ϵ. However, given the large action-state space, a randomly exploring Q-learner is not sample efficient, wherein convergence to an optimal policy is slow and successful dialogues are hard to achieve. The Q function value of each state-action pair can be augmented with an estimate of its uncertainty to guide exploration, and achieve faster learning and a higher reward during learning.

Gaussian Processes (GPs) provide an explicit estimate of uncertainty and currently, GP-SARSA, which uses GPs to approximate the Q-function, constitutes the state of the art for policy optimization in dialogue management [12]. However, GPSARSA requires computing the inverse of a Gram matrix K for determining the predictive posterior and estimating Q at new locations. In a dialogue system, the total number of turns needed to accurately estimate the optimal policy is large, and if every datapoint is memorized, the computational complexity increases in a cubic order, because of the matrix inversion. Sparcification methods are then needed to achieve lower computational complexity. This may limit however the accuracy of the solution.

Deep neural network models on the other hand scale well with data and are computationally less complex than GPs. They do not provide an estimate of uncertainty, however, and current deep RL methods rely on ε-greedy exploration, and are not as sample efficient as GPSARSA. This thesis focus on extracting uncertainty information in deep Q-networks to increase sample efficiency. This is done via the Bayes-By-Backprop method, where a probability distribution over the weights in the network is utilized.

0.2 Contributions

This thesis implements and compares the Bayes By Backprop method for deep-Q-networks (BBQN) to GP-SARSA, and other deep RL methods, and investigates how BBQN can be improved and be made competitive with GP-SARSA. We implement BBQN in the Cambridge University Engineering Department dialogue systems toolkit, CUED-pydial, and compare its performance to GP-SARSA, deep Q-networks (DQN) and deep policy based methods such as the episodic Natural Actor-Critic. We show that BBQN learns dialogue policies with more efficient exploration than ε-greedy based methods, and reaches performance comparable to the state of the art in policy optimization, namely GPSARSA, especially when evaluated on more complex domains.

We also implement α-divergences, variational dropout, and minimizing the negative log likelihood as other means to extract uncertainty estimates from DQN, and compare performance to BBQN and DQN.

Chapter 1

Overview of Spoken Dialogue Systems

The principle elements of the Spoken Dialogue System (SDS) used for this research work are illustrated in Fig. 1.1. One cycle through the system corresponds to one turn in a dialogue. The system is task oriented, where the domain is presented using slots, which are variables the user can either specify or ask about in the domain.

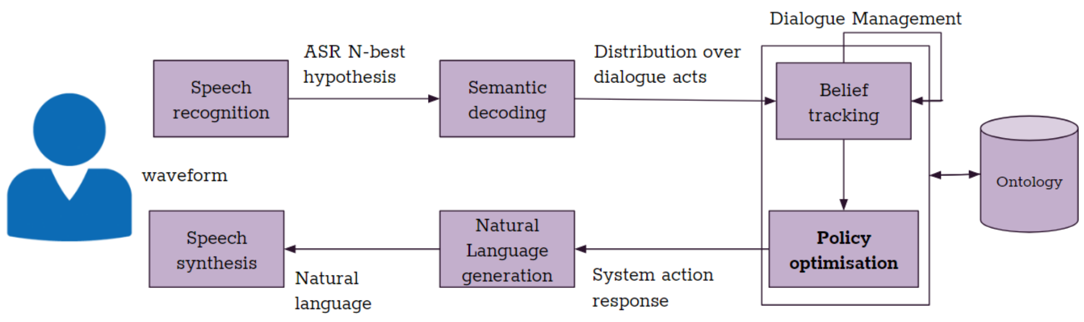


Figure 1.1: Components of a modular spoken dialogue system.

In spoken dialogue systems, automatic speech recognition (ASR) and spoken language understanding (SLU) technology are error-prone, and user behavior is unpredictable. A robust SDS is one that implements strategies that account for the unreliability of the input and provides error checking and recovery mechanisms [51]. A sequential and modular approach allows the implementation of the POMDP framework. This approach, which is detailed in section 2.1.2, regards the outputs of the ASR and the SLU components as a noisy observation of the user input. It uses a stochastic model, effectively dealing with an increasing uncertainty with speech recognition and language understanding [49]. The sequential approach minimizes dependence on explicit rules as well. A summary of each component is presented below.

1.1 Automatic Speech Recognition

The ASR component outputs a posterior probability to the words of an utterance given its acoustics. Instead of retaining the most probable hypothesis only, an N-best list of hypotheses with corresponding probabilities is often used for the output [17]. An N-best list allows to retain a potentially correct hypothesis that was ranked less probable because of imperfect speech recognition, where errors can be present in the transcribed speech especially in noisy environments.

1.2 Spoken Language Understanding

The task of the SLU component or semantic decoder is to discern the semantics of an utterance given the output of the ASR. To capture meaning in an utterance, the semantic decoder takes a sentence as input and gives a dialogue act as output [17]. A dialogue act consists of a dialogue act type, (e.g. requesting information, informing constraints, saying good-bye), and a slot-value pair that specify arguments of the act. For example `inform(food=chinese)` is a dialogue act of type inform, where the user is informing the system that they would like to constrain their search to venues serving Chinese food [17].

Parse trees with hand-written rules that map words and phrases to semantic concepts can be used for the semantic decoder. Recent work has focused on end-to-end memory networks where utterances are encoded with intents, and slots are stored as embeddings in the memory [3]. The decoding phase applies an attention model to leverage previously stored semantics for intent prediction and slot tagging simultaneously, and knowledge from previous turns is carried into the current turn. This, such model addresses errors from previous dialogue turns that can propagate and degrade the performance of the current turn.

1.3 Dialogue State Tracking

Spoken dialog systems need to keep a representation of the dialog state and the user goal to follow an efficient interaction path [21]. Dialog state tracking methods need to cope with error-prone ASR and SLU outputs. For each turn of the dialogue, given a sequence of SLU hypotheses from the beginning of the dialog up to the turn t, a state tracker outputs the probability distribution for each of the three components of the dialogue state:

1. the user's hidden goals (the user's value for a given slot, also called informable slot)
2. the methods (the way the user is trying to interact with the system),
3. the requested slots (information the user is asking for).

For this thesis, the baseline focus tracker is used. Compared to a one best tracker, a focus tracker makes use of the full distribution given by the SLU, accounts well for goal constraint changes; and accumulates evidence from the SLU over time. Evidence is integrated across turns, reducing the impact of errors and rewarding user persistence [51].

The focus tacker updates a value $q_{c,t}$, which denotes the probability the SLU did not inform a method and a goal component c at turn t, as follow:

$$q_{c,t} = 1 - \sum_{v \in V_c} slu_{t,c}(v) \tag{1.1}$$

$$p_{c,t}(v) = slu_{t,c}(v) + q_{c,t} p_{c,t-1}(v) \tag{1.2}$$

$$p_{c,1}(v) = slu_{1,c}(v), \tag{1.3}$$

with $p_{c,t}$, the probability distribution for a component of the dialogue state, and v, the value for the component c. If there is no evidence that c should be reset in the SLU, then $q_{c,t} = 1$ and $p_{c,t}$=$p_{c,t-1}$. The more evidence there is, the smaller $q_{c,t}$ is. If the context is sure that c is to be reset, then $q_{c,t} = 0$ and $p_{c,t} = slu_{t,c}(v)$.

As an alternative to the focus tracker, Henderson [17] suggests a word-based tracking method that maps directly from the speech recognition results to the dialogue state without using an explicit semantic decoder. The method is based on a recurrent neural network structure that is capable of generalising to unseen dialogue state hypotheses, and requires very little feature engineering.

1.4 Ontology

The ontology defines the domain of the dialogue system. It includes a list of requestable and informable slots and their values, and a list of entities on which queries can be performed. These two components define the actions the system can take, and the tasks the system can fulfill.

1.5 Dialogue Management

After the dialogue state has been estimated, the dialogue system needs to respond appropriately. This is the task of the dialogue management component, which is the focus of this thesis, and is discussed in further details in chapter 2.

1.6 Natural Language Generation and Speech Synthesis

The last component of a SDS consists of generating text corresponding to the system's chosen dialogue act and synthesizing the text into audio. One method to generate a system's chosen dialogue act is to search for the most likely sequence of semantic concepts and words using Factored Language Models [29]. For synthesizing speech from text, WaveNet constitutes the state of the art [34]. WaveNet is a probabilistic and autoregressive model, with the predictive distribution for each audio sample conditioned on all previous ones.

Chapter 2

Dialogue Management

2.1 Policy Optimization

2.1.1 The Markov Decision Process

Traditional approaches to dialogue management have used hand-crafted methods, which do not learn which actions should be taken. Instead, the dialogue management can be modeled as a Markov Decision Process (MDP), where the dialogue is represented as a sequence of states s_t. Each state encapsulates enough information about the dialogue up to turn t so that the model satisfies the Markov property. In each state, the system takes an action a_t, moves to a new state s_{t+1} and receives a reward r_t. The transition to any state depends only on the previous state and the action that was taken [51, 12].

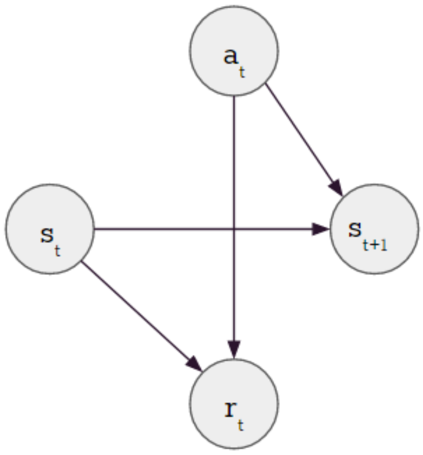


Figure 2.1: Diagram of a MDP.

2.1.2 POMDP-based Dialogue Manager

The MDP model assumes the dialogue state correctly represents the user input. This is not the case however, since uncertainty is introduced due to errors carried by the ASR and the SLU components of the SDS. The Partially Observable Markov Decision Process (POMDP) intrinsically deals with this uncertainty and is used to model dialogue. POMDP assumes the dialogue state is only partially observable and depends on a noisy observation o_t, defined by an observation probability $P(o_t|s_t)$ [12]. The observation vector is derived from a combination of the user's dialogue acts and the system state, and encapsulates the evidence needed to update and maintain the system beliefs as accurately as possible [50].

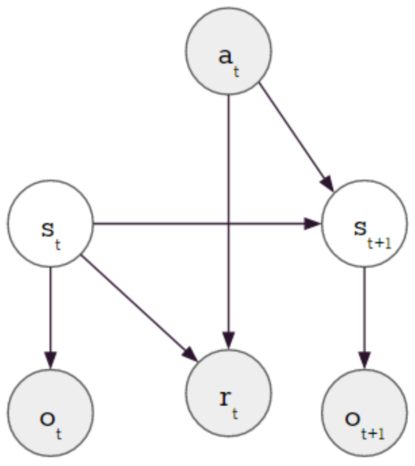


Figure 2.2: Diagram of a POMDP. Clear circles describe hidden random variables and shaded circles are observed random variables

A POMDP is a tuple ($\mathcal{S}$, $\mathcal{A}$, P_T, $\mathcal{R}$, $\mathcal{O}$, P_O, γ, b_0) where:

- $\mathcal{S}$ is the state space.
- $\mathcal{A}$ is the action space.
- $P_T(s, s_0) = P_T(s_{t+1}|s_t, a_t)$, is the function encoding the transition probability from state s_t to s_{t+1}, after taking action a_t.
- $\mathcal{R}$ is the reward function with $r(s, a) = R_{success}$ for action-state pairs that directly lead to a successful dialogue and $r(s, a) = -R_{penalty}$ for all other actions. The reward function encourages short dialogues which provide the information requested by the user.
- O is the observation space which is the output of the SLU component.
- $P_O(o_t|s_t, a_t)$ defines the observation probabilities, which encodes how hidden states lead to the observed states.
- γ is a geometric discount factor with $0 \leq \gamma \leq 1$. Lower values of γ further penalizes future reward.

- b_0 is an initial belief state.

Starting from b_0, the state tracker maintains and updates the belief state according to the observations perceived during the dialogue [7]. The dialogue manager operates on this belief state, with $b_t(s_t)$ indicating the probability of being in a particular state s_t [51]. Based on b_t, an action a_t is selected, a reward r_t is received, and the system transitions to (unobserved) state s_{t+1}, which depends only on s_t and a_t. Updating the exact belief state requires a summation over all states, which is intractable for very large state spaces [22]:

$$b_{t+1}(s_{t+1}) \propto P(o_{t+1}|s_{t+1}, a_t) \sum_{s \in \mathcal{S}} P(s_{t+1}|s_t, a_t) b_t(s_t) \tag{2.1}$$

If an approximation of the POMDP belief state can be tractably maintained at every dialogue step, the POMDP with discrete state and observations spaces can be treated as a MDP with continuous states [22], and MDP algorithms with parametrization can be used for policy optimization. When an action is selected based on the differing levels of uncertainty in each dialogue state as well as the overall reward, a distribution of states (belief state) is maintained at each turn. This explicit representation of uncertainty in the POMDP gives it the potential to produce more robust dialogue policies [51, 12].

2.2 Reinforcement Learning

Given a belief state, selecting an action that maximizes the long term objective can be solved with reinforcement learning (RL). The RL problem consists of an environment (the user) and an agent (the system) [44] and the goal is to find an optimal policy $\pi*$, which determines the system actions, and which maximizes the discounted total return:

$$R = \sum_{t=0}^{T} \gamma^t r_t(b_t, a_t) \tag{2.2}$$

over a dialogue with T turns. $r_t(b_t, a_t)$ is the reward when the dialogue is in belief state b_t and when action a_t is taken. γ is a geometrical discount factor and a large value indicates the agent is far sighted, and a value close to 0 indicates the agent is short sighted where future rewards are not taken into consideration.

The expected discounted sum of rewards for a belief state b_t and following policy π is given by the value function $V^\pi(b_t)$. With a deterministic policy, the value function can be expressed as [44]:

$$V^\pi(b_t) = r\left(b_t, a_t\right) + \gamma \sum_{o_{t+1}} P\left(o_{t+1}|b_t, a_t\right) V^\pi(b_{t+1}) \tag{2.3}$$

Given a belief state and action pair, a quantity called Q-function is defined where

$$V^\pi(b) = Q^\pi(b, a). \tag{2.4}$$

Given a POMDP, learning an optimal policy can be reduced to learning an optimal value function $Q*$, where the optimal action corresponds to $a* = arg \max_a Q*(b, a)$.

Policy optimization in dialogue systems is resolved with model-free methods that do not explicitly learn a system model, but rather use sample trajectories obtained by direct interaction with the system [15]. Model free methods include SARSA and Q-learning. Finding the optimal policy can also be solved by either value-based or policy-based methods. Value function methods find the optimal value function then extract an optimal policy from it. Such methods include policy iteration, value iteration, Q-leaning, SARSA. Policy-based methods directly search in the space of policies. However, the number of policies is exponential in the size of the state space and local greedy methods must be employed [15]. Policy gradient methods can be employed instead, where the policy is taken as a differentiable function of a parameter function, and the search in the policy space is directed by the gradient of the function with respect to the policy parameters.

2.2.1 Exploration vs Exploration

In a spoken dialogue system, the agent does not know transition probabilities and rewards, and has to trade-off exploration and exploitation to learn an optimal policy. Exploitation corresponds to the agent exploiting what it already knows, i.e. selecting actions that it has tried in the past and found to be effective in producing reward [44]. Exploration on the other hand corresponds to the agent exploring the environment, to make better action selections in the future. Exploration requires the agent to try actions it has not selected before. Neither exploration nor exploitation can be pursued exclusively without failing at the task. In tasks with a large action-state space, the agent will not reach an optimal solution with too much exploration, and too much exploitation can only lead to sub-optimal solutions.

2.2.2 ε-greedy Exploration

To explore the environment, an ε-greedy policy can be employed, where a greedy action is taken with respect to the estimated Q function ($arg\max_a Q(b,a)$) with probability $1-\varepsilon$, and a random one with probability ϵ [5]. As the number of turn increases, every action is sampled an infinite number of times, ensuring that all the $Q_t(a)$ converge to $Q^*(a)$. This implies that the probability of selecting the optimal action converges to greater than $1-\varepsilon$, that is, to near certainty. This is just an asymptotic guarantee, however, and in practice, the method is ineffective, especially for tasks with a large action-state space [44]. In the next section, alternatives to ε-greedy exploration are explored.

2.2.3 SARSA

SARSA is a Temporal Difference (TD) learning method. TD methods can learn directly from raw experience without a model of the environment's dynamics. SARSA is a model-free method for policy evaluation that bootstraps the value function from subsequent estimates. In SARSA, the value function is bootstrapped from the very next time-step. Rather than waiting until the complete return has been observed (as with Monte Carlo methods), the value function of the next state is used to approximate the expected return [44]. SARSA estimates an action value function, Q(b, a), for each state s and action a. At each time-step t, the next action a_t is selected by an ε-greedy policy with respect to the current action values Q(b,·). The Q function is updated as:

$$Q(b_t, a_t) = Q(b_t, a_t) + \alpha[r_{t+1} + \gamma Q(b_{t+1}, a_{t+1}) - Q(b_t, a_t)]. \tag{2.5}$$

Here α is the learning rate which determines how much attention is given to new experiences. This update is done after every transition from a nonterminal belief state b_t. If b_{t+1} is terminal, then $Q(b_{t+1}, a_{t+1})$ is defined as zero. If all states are visited infinitely many times, and ε decays to zero in the limit, SARSA converges to the optimal action-value function Q^* for all values of λ [44].

2.2.4 Q-Learning

In Q-Learning, the learned action-value function, Q, directly approximates $Q*$, the optimal action-value function using a target policy, independent of the policy being followed - the behavior policy. The update rule for the Q function is as follow:

$$Q(b_t, a_t) = Q(b_t, a_t) + \alpha(r_{t+1} + \gamma max_{a'} Q(b_{t+1}, a') - Q(b_t, a_t))) \tag{2.6}$$

The behavior policy has an effect in that it determines which state–action pairs are visited and updated and is determined epsilon-greedily, ensuring sufficient exploration, and all that is required for correct convergence is that all pairs continue to be updated [44].

2.2.5 Non-Parametric Policy Modeling

Exact policy optimization for POMDPs is intractable for everything except for very simple cases [22]. One can discretize the belief state and apply a tabular approach to obtain the Q value for a state-action pair. However, the number of states, actions and observations is very large in most real-world applications, including spoken dialogue systems, and representing the value function as a a table only leads to poor or sub-optimal solutions.

One way to mitigate this issue is to approximate the belief state. One approach is the hidden information state (HIS) model which maps the belief state space into a smaller grid-based discretized summary belief space [52]. Using a summary belief space allows faster convergence to a solution, however, the speed of convergence and the quality of the approximation depend on the number of grid points used [12]. A small number of grid points can lead to faster convergence but to suboptimal results, since too many very different belief state points are approximated into the closest grid point. With a large number of grid points, optimizing the policy becomes intractable. Further, 10^5 dialogues are still required to train a HIS system, which is only possible with a user simulator [51].

Instead, approximations that provide compact representations for the model and the policy can be used to allow tractable algorithms for performing belief monitoring and policy optimization [51]. As mentioned above, the POMDP can be approximated as a continuous state MDP when tractably maintaining an approximation of the belief state.

For instance, the Q-function can be approximated with parametric models. An example of such models is to represent the Q-function as a set of basis functions with $Q_\theta(b, a) = \theta^T \phi(b, a)$ [4]. Here, $\phi(b, a)$ is a feature vector of size N (number of parameters) so that $\phi = [\phi_1(b, a)...\phi_N(b, a)]^T$ is a predefined set of basis functions and θ are the associated weights.

Non-parametric policy modeling avoids, on the other hand, the limitations of constraining the solution to basis functions or the number of grid points for a summary belief state. A non-parametric representation requires still parameters, however the choice of these parameters do not restrict the solution, but rather affect the speed at which the optimal solution is found [12]. Non-parametric models include Gaussian processes which are discussed in details in the next sections.

2.3 Learning Under Uncertainty

Efficient exploration is key to successful dialogue policy estimation. Given a Q-function approximation algorithm, choosing actions according to the current Q function estimate requires a choice between exploration and exploitation [5].

To explore the environment, an ε-greedy policy can be employed. However, for dialogue systems, the reward is sparse, and the action and state space is large. A dialogue manager should also improve quickly and adapt rapidly to users, and learning should be safe (bad actions should not repetitively be chosen). Under these conditions, a randomly exploring Q-learner fails.

The dialogue manager can instead make use of an estimate of uncertainty of the Q function value of each state-action pair to guide exploration. The dialogue manager chooses the action which leads to the highest reward when it is confident about its estimate of that reward, otherwise it will choose the action about which it is least certain [5]. This reduces the frequency of choosing bad actions, and faster and more efficient learning is achieved.

2.3.1 Thompson Sampling

Given a posterior distribution approximating the Q function, an agent can sample a value function from this posterior at every time step and can choose the action which is optimal for that time step. This approach is called Thompson Sampling [46] and is a popular means of picking an action that trade-offs between exploitation and exploration, effectively making use of uncertainty estimates provided by the posterior distribution. More probable parameters are drawn more often and thus refuted or confirmed the fastest [2]. Thompson sampling works as follow [2]:

Algorithm 1 Thompson Sampling

```
1: Sample a new set of parameters for the model.
2: Pick the action with the highest expected reward according to the sampled parameters.
3: Update the model.
```

2.4 Gaussian Processes

Currently, Gaussian Process SARSA is the state of the art algorithm for policy optimization in POMDP-based dialogue systems [12, 43]. A Gaussian process is a non-parametric Bayesian model used for function approximation [12], and is defined by a distribution over functions with a prior mean function and a covariance function [38]:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

with

$$\begin{aligned} m(x) &= \mathbb{E}(f(x)) \\ k(x, x') &= \mathbb{E}[(f(x) - m(x))(x' - f(x'))]. \end{aligned}$$

A Gaussian process prior distribution on f(x) allows to encode assumptions about the smoothness (or other properties) of the latent function. The kernel $k(x, x')$ determines how the similarity between a pair of function values varies as a function of the corresponding pair of inputs, and depends on a small number of hyperparameters [37].

A key assumption in GP is that data can be represented as a sample from a multivariate Gaussian distribution with

$$\begin{bmatrix} y \\ y_* \end{bmatrix} \sim \mathcal{N}(0, \begin{bmatrix} K & K_*^T \\ K_* & K_{**} \end{bmatrix}) \tag{2.7}$$

where y_* is a prediction and y is an observation.

The conditional probability $p(y^*|y)$ follows a Gaussian distribution:

$$y_*|y \sim \mathcal{N}(K_*K^{-1}y, K_* * -K_*K^-1K_*^T) \tag{2.8}$$

where K denotes the covariances between two points as follow:

$$K_* = [k(x_*, x_1) \ ... \ k(x_*, , x_n)] \tag{2.9}$$

$$K_{**} = [k(x_*, x_*)] \tag{2.10}$$

The best estimate for y_* is the mean of the conditional distribution:

$$\bar{y}_* = K_*K^{-1}y,$$

and the variance of this distribution captures the uncertainty about the estimate:

$$var(y_*) = K_{**} - K_*K^{-1}K_*^T$$

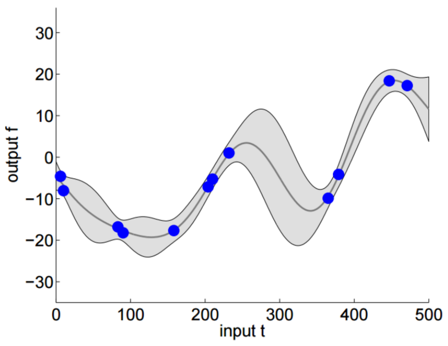


Figure 2.3: Posterior distribution over an unknown function as produced by a GP. The evidence is represented by blue dots. The gray line represents the mean of the posterior distribution and the gradient indicates the variance up to two standard deviations.

2.4.1 GP-SARSA

GPs can be used to model the Q-function. The discounted return at time step t is the total reward acquired using policy π starting from time step t. This can be written as:

$$R_t = \sum_{i=0}^{\inf} \gamma^i r_{t+i+1} \tag{2.11}$$

$$= r_{t+1} + \gamma R_{t+1}, \tag{2.12}$$

where r_t is the immediate reward at time step t and γ is the discount factor. If the immediate reward is a random process, so does the discounted reward [12]. The Q-function of a policy π is the expectation of the discounted return for that policy given belief state b and action a at time t:

$$Q(b, a) = \mathbb{E}[R_t | b_t = b, a_t = a]. \tag{2.13}$$

During the process of estimation, $Q(b, a)$ can be considered as a random value and the return at time t can be decomposed into a mean $Q(b, a)$ and a residual $\Delta Q(b, a)$ [12]:

$$R_t(b_t = b, a_t = a) = Q(b, a) + \Delta Q(b, a) \tag{2.14}$$

Substituting 2.14 into 2.11, one can express the immediate reward using the Q-value:

$$r_{t+1}(b, a) = Q(b, a) - \gamma Q(b', a') + \Delta Q(b, a) - \gamma \Delta Q(b', a') \tag{2.15}$$

In GPSARSA, the unknown Q-function is modeled as a Gaussian Process with the following prior:

$$Q(b, a) \sim \mathcal{GP}(0, k((b, a), (b, a))), \tag{2.16}$$

where the kernel is factored into separate kernels over the belief state and action spaces [14]:

$$k_B(b, b') k_A(a, a') \tag{2.17}$$

The prior for the residual follows $\Delta Q(b, a) \sim \mathcal{N}(0, \sigma^2)$. Gaussian process regression is then applied to find the posterior of $Q(b, a)$, given a set of belief state-action pairs and the observed rewards. Initially, the estimated Q-function distribution is a zero-mean Gaussian process with the kernel function as in 2.17, since no data has been observed. As training progresses, updates take place every time a reward is observed, and at turn t, the estimate is the posterior distribution given the set of observed rewards r_t and the belief state action pairs [12]. Thompson sampling is then used to get an estimate of the Q value (instead of choosing the mean of the distribution).

By describing correlations in different parts of the belief state space through the kernel function, GPs incorporate knowledge of dependencies between belief states [12]. In addition, GPs provide an uncertainty about the estimate and a sampling based approach allows efficient exploration.

Sparse approximation

Computing the predictive posterior and estimating Q at new locations requires computing the inverse of the covariance matrix K. In a dialogue system, the number of points used for estimation is equal to the total number of turns, summed over all dialogues [14], and the total number of turns needed to accurately estimate the optimal policy is large. If every datapoint is memorized, the computational complexity increases of the order t^3 where t is the number of data points, because of the matrix inversion. Sparcification is employed to reduce the computational complexity while taking into account the contribution each part of the belief state space brings to the final estimation.

With GP-SARSA, which is an online learning method, support points cannot be preselected, and the kernel span sparsification method is employed instead. This technique consists of approximating the kernel on a dictionary of linearly independent belief states. The dictionary is incrementally built during learning. A threshold is required on the precision to which the kernel is computed and is fine-tuned for a good tradeoff between precision and performance.

A point (b^t, a^t) is added to the dictionary if the approximate linear dependence (ALD) condition is satisfied:

$$\min_{g_t} \left\| \sum_{j=0}^{m} g_{t,j}\phi(\tilde{b}^j, \tilde{a}^j) - \phi(b^t, a^t) \right\|^2 \geq \nu. \tag{2.18}$$

This sparcification reduces complexity of calculating the posterior form from $O(t^3)$ to $O(tm^2)$, where m is the number of representative points.

Kernel Functions

For the belief state space, the linear kernel function is used in the GPSARSA implementation of the CUED-PyDial toolkit:

$$k((b,a),(b',a')) = \langle b, b' \rangle \delta_a(a') \tag{2.19}$$

The linear kernel assumes the elements of the space are features. The Kronecker delta function $\delta_a(a')$, is used for the summary action space since it is discrete. The covariance between (b, a) pairs is retained only when the two actions are the same.

Algorithm 2 provides a summary of GPSARSA.

Algorithm 2 GPSARSA

```
 1: Define prior for Q-function
 2: for each dialogue do
 3:   Initialize $b$ and choose $a$ according to current Q estimate
 4:   if $(b, a)$ is representative then
 5:     add pair to dictionary
 6:   end if
 7:   for each turn do
 8:     Take action $a_t$, observe $r_t$ and next belief state $b_{t+1}$
 9:     Choose $a_{t+1}$ according to current $Q$ estimate
10:     if $(b_{t+1}, a_{t+1})$ is representative then
11:       add pair to dictionary
12:     end if
13:     Update posterior mean and variance of Q
14:     $b_{t+1} \to b_t$, $a_{t+1} \to a_t$
15:   end for
16: end for
```

Chapter 3

Deep Reinforcement Learning

Instead of using GPs, one can use deep neural networks (DNNs) to obtain an approximation of the Q-function. Deep neural networks scale well with data, and computational complexity does not increase with increasing number of dialogues during training.

Neural networks can be viewed as a probabilistic model $P(y|x, w)$. Given an input x, the neural network assigns a probability to the output $y \in \mathcal{Y}$ using the set of parameters or weights w [2]. Neural networks are defined by applying consecutive transformations, often applying linear transformations followed by nonlinear ones. For instance, a two layer neural network can be represented by:

$$y = \sigma_2(W_2\sigma_1(W_1x + b_1) + b_2), \tag{3.1}$$

where $\sigma_1(\cdot)$ and $\sigma_2(\cdot)$ denote activation functions, and W_1, W_2, b_1, and b_2 denote matrices and vectors of parameters corresponding to the linear transformations. For a regression task, $\mathcal{Y}$ is $\mathbb{R}$ and the last layer outputs a probability distribution. For a classification task, $\mathcal{Y}$ is a set of classes, and $P(y|x, w)$ is a categorical distribution obtained through a softmax transformation of the output of the last layer.

Combining deep neural networks with RL benefits from an efficient generalization over the belief space, with considerably less computational time for training than GPSARSA. However, DNNs do not provide an estimate of uncertainty, which is key for efficient exploration. This section details how deep neural networks are used in RL, and methods for extracting uncertainty estimates.

3.1 Deep Q-Learning

In deep Q-learning, a deep neural network is used to parametrize the Q function. A Deep Q-Network (DQN) maps a belief state b_t to the values of the possible actions a_t at that state, $Q(b_t, a_t; w_t)$ [7]. w_t designates the weight vector of the neural network. The optimal value function is found by minimizing the squared error between the current prediction and the one-step look-ahead:

$$\mathcal{L}(w_t) = \mathbb{E}[(y_t - Q(b_t, a_t; w_t))^2], \tag{3.2}$$

where $y_t = r_t + \gamma \max_{a'} Q(b_{t+1}, a'; w_t)$ is the target to update the parameters w. The network's weights are updated using gradient descent:

$$w_{t+1} \leftarrow w_t + \alpha(y_t - Q(b_t, a_t; w_t))\nabla Q(b_t, a_t; w_t). \tag{3.3}$$

In online RL, the agent incrementally update the parameters. Learning directly from consecutive samples produce strongly correlated samples. To mitigate this issue, we use experience replay, which maintains a buffer of experiences [40]. Mini-batches are randomly drawn from this buffer, which reduces the temporal correlations between updates, smooths out learning and avoids oscillations or divergence in the parameters.

A separate network for generating the targets y_t in the Q-learning update is also used. The parameters of the target network $\hat{Q}$ are updated using the network Q after C updates. An alternative is to update the $\hat{Q}$ network continuously but slowly. This modification makes Q-learning more stable, because an update that increases $Q(b_t, a_t)$ often also increases $Q(b_{t+1}, a)$ for all a, which increases the target y_t leading to oscillations or divergence of the policy [31]. Using a target network adds a delay between the time an update to Q and the time the update affects the target y_t. In addition to the previous modifications, the error term from the update is clipped to be between -1 and 1, which can further improve the stability of the algorithm.

Other techniques have been employed to further improve learning with DQN, such as double DQN [48] and prioritized experience replay [40], however none of these were investigated in this thesis.

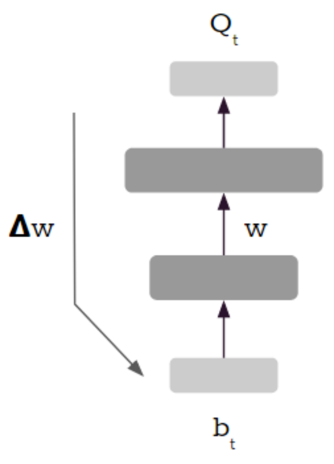


Figure 3.1: DQN architecture using a feed-forward neural network. Δw designates a Q-value gradient, Q_t is a Q scalar, b_t is a dialogue state.

3.1.1 Exploration

The ε-greedy exploration method is used for DQN. More efficient exploration can be achieved, however, with uncertainty estimates. In this thesis, we implement the Bayes-By-Backprop method for extracting uncertainty estimates from DQN, and explore other methods, such as dropout or Bayes By Backprop with α-divergences, in attempt to improve upon the ε-greedy exploration. We introduce these methods in the next sections.

DQN with experience replay and a target network is summarized in Algorithm 3.

Algorithm 3 DQN with experience replay and a target network

```
1: Initialize replay memory $D$ to capacity $N$
2: Initialize action-value function $Q$ with random weights $w_t$
3: Initialize target action-value function $\hat{Q}$ with $w_t^- = w_t$
4: for dialogue = 1, M do
5:   Initialize $b_1$
6:   for turn t = 1,T do
7:     Select random action $a_t$ with probability $\varepsilon$, otherwise, select $a_t = arg\max_a Q(b_t, a; \theta)$
8:     Take action $a_t$, observe $r_t$ and $b_{t+1}$
9:     Store transition $(b_t, a_t, r_t, b_{t+1})$ in $D$
10:    Sample random minibatch of transition $(b_j, a_j, r_j, b_{j+1})$ from D
11:    if Episode terminates at step j+1 then
12:      Set $y_j = r_j$
13:    else
14:      $y_j = r_j + \gamma max_{a'} \hat{Q}(b_{j+1}, a'; w^-)$
15:    end if
16:    Perform a gradient descent step on $(y_j - Q(b_j, a_j; w))^2$ with respect to the network
       parameters $w$
17:    Set $\hat{Q} = Q$ every C steps
18:  end for
19: end for
```

3.2 Measuring Uncertainty in Deep Neural Networks

To obtain uncertainty estimates from a neural network, Bayesian neural networks (BNNs) can be employed [32, 28]. BNNs are standard neural networks with prior probability distributions placed over the weights. Inference however is difficult, but variational inference techniques such as sampling-based variational inference and stochastic variational inference [19, 1, 16] can be applied. Using these techniques, a simple approximate learning algorithm called Bayes By Backpropagation [2] is obtained.

3.2.1 Bayes By Backpropagation

Instead of having single fixed value weights in the neural networks, all weights are represented by probability distributions over possible values. Uncertainty in the hidden units allows the expression of uncertainty about an observation [2]. The posterior distribution of the weights given the training data, $P(w|\mathcal{D})$, is calculated with Bayesian inference. The output $\hat{y}$ given the input test data item $\hat{x}$ is given by:

$$P(\hat{y}, \hat{x}) = \mathop{\mathbb{E}}_{P(w|\mathcal{D})}[P(\hat{y}|\hat{x}, w)]. \tag{3.4}$$

Taking an expectation under the posterior distribution is equivalent to using an ensemble of an uncountably infinite number of neural networks, which is intractable [2]. The intractable posterior is approximated with a variational distribution $q(w|\theta)$. The parameters are learnt by minimizing the Kullback-Lieber ($\mathcal{KL}$) divergence between the variational approximation $q(w|\theta)$ and the true posterior on the weights $P(w|\mathcal{D})$:

$$\begin{aligned}
\theta &= arg\min_{\theta} \mathcal{KL}[q(w|\theta)||P(w|\mathcal{D})] \\
&= arg\min_{\theta} \int q(w|\theta) \log \frac{q(w|\theta)}{P(w)P(\mathcal{D}|w)} dw \\
&= arg\min_{\theta} \int q(w|\theta) \log q(w|\theta) dw - \int q(w|\theta) \log P(w)P(\mathcal{D}|w) dw \\
&= arg\min_{\theta} \mathcal{KL}[q(w|\theta)||P(w)] - \mathop{\mathbb{E}}_{q(w|\theta)}[\log P(\mathcal{D}|w)].
\end{aligned}$$

The resulting cost function is termed as the variational free energy [19]:

$$\mathcal{F} = \mathcal{KL}[q(w|\theta)||P(w)] - \mathop{\mathbb{E}}_{q(w|\theta)}[\ln P(\mathcal{D}|w)]. \tag{3.5}$$

This cost function includes a data-dependent part, namely the likelihood cost or expected log likelihood, and a prior-dependent part called the complexity part, which also serves as regularization. Maximizing the expected log likelihood encourages $q(w|\theta)$ to explain the data well, while minimizing the prior encourages $q(w|\theta)$ to be as close as possible to the prior. This procedure is known as variational inference (VI), a standard technique in Bayesian modeling. Variational inference replaces the Bayesian modeling marginalization with optimization, where the calculation of integrals is replaced with that of derivatives.

However, minimizing the exact cost is computationally prohibitive, mainly because of the integral in the expected log likelihood which cannot be evaluated analytically [2], and approximations and gradient descent are used.

3.2.2 The reparameterization trick

A Monte Carlo gradient estimator for $\nabla_\theta \, \mathbb{E}_{q(w|\theta)}[\ln P(\mathcal{D}|w)]$ exhibits very high variance [23], and when the variance of the gradients is too large, stochastic gradient descent may fail. To propagate the error through a layer that samples from $q(w|\theta)$, the reparameterization trick [23] is used. As outlined in [2], let ϵ be a random variable with probability density $q(\epsilon)$ and let $w = t(\theta, \epsilon)$ where $t(\theta, \epsilon)$ is a deterministic function. If $q(\epsilon)d\epsilon = q(w|\theta)dw$, then for a function f with derivatives in w, we have:

$$\begin{aligned}
\frac{\partial}{\partial\theta} \mathop{\mathbb{E}}_{q(w|\theta)}[f(w,\theta)] &= \frac{\partial}{\partial\theta} \int f(w,\theta) q(w|\theta) dw \qquad (3.6) \\
&= \frac{\partial}{\partial\theta} \int f(w,\theta) q(\epsilon) d\epsilon \qquad (3.7) \\
&= \mathop{\mathbb{E}}_{q(\epsilon)}[\frac{\partial f(w,\theta)}{\partial w} \frac{\partial w}{\partial\theta} + \frac{\partial f(w,\theta)}{\partial\theta}]. \qquad (3.8)
\end{aligned}$$

We choose $q(w|\theta)$ to be a Gaussian with diagonal covariance. Given the mean μ_i and covariance σ_i of q for each weight, a sample from q is obtained by first sampling $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon)$, then computing $w_i = \mu_i + \sigma_i \circ \epsilon_i$, where $\circ$ is point-wise multipliation. To ensure all σ_i are strictly positive, the softplus function $\sigma_i = \log(1 + \exp(\rho_i))$ is used [27]. The variational parameters are then $\theta = \{\mu_i, \rho_i\}_{i=1}^{\mathcal{D}}$ for D-dimensional weight vector w.

Removing the dependence on sampled parameters θ of posterior samples w_i, allows ∇_θ to pass through the expectation symbol in 3.5 and compute derivatives with respect to the network parameters [9]. The resulting gradient estimator of the variational objective is unbiased and has a lower variance. The exact cost in 3.5 can then be approximated as:

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^{n} \log q(w^{(i)}|\theta) - \log P(w^{(i)}) - \log p(\mathcal{D}|w^{(i)}) \tag{3.9}$$

where $w^{(i)}$ is the ith Monte Carlo sample drawn from the variational posterior $q(w^{(i)}|\theta)$. With minibathes drawn from the buffer, the minibatch cost for minibatch i = 1, 2, ..., M is:

$$\mathcal{F}_i(\mathcal{D}_i, \theta) = \xi_i \mathcal{KL}[q(w|\theta)||P(w)] - \mathop{\mathbb{E}}_{q(w|\theta)}[\ln P(\mathcal{D}_i|w)] \tag{3.10}$$

where ξ_i is the weight for the KL cost. Each step of the optimization is summarized in Algorithm 4.

Algorithm 4 Optimisation in Bayes By Backprop for deep neural networks

```
1: Sample $\epsilon \sim \mathcal{N}(0, I)$ which yields a differentiable MC approximation.
2: Let $w = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
3: Let $\theta = (\mu, \rho)$ and $f(w, \theta) = \xi_i(\log q(w|\theta) - \log P(w)) - \log P(\mathcal{D}|w)$.
4: Calculate the gradient with respect to the mean $\mu$, and the standard deviation parameter $\rho$:
5: $\Delta_\mu = \frac{\partial f(w,\theta)}{\partial w} + \frac{\partial f(w,\theta)}{\partial \mu}$
6: $\Delta_\rho = \frac{\partial f(w,\theta)}{\partial w} \frac{\epsilon}{1+\exp(-\rho)} + \frac{\partial f(w,\theta)}{\partial \rho}$
7: Update the variational parameters:
8: $\mu \leftarrow \mu - \alpha\Delta_\mu$
9: $\rho \leftarrow \rho - \alpha\Delta_\rho$
```

3.2.3 Thompson Sampling

For efficient exploration, Thompson sampling is used, where at each turn, a new set of parameters is drawn and an action is picked relative to those parameters. At the beginning of training, the actions are picked uniformly, as the variational posterior is close to the prior $P(w)$. q will start to converge as the training proceeds, and uncertainty on the parameters will decrease. Action selection becomes more deterministic, focusing on the high expected reward actions discovered so far [2]. The algorithm is summarized in Algorithm 5 [2]:

Algorithm 5 Thompson Sampling in Bayesian Neural Networks

```
1: Sample weights from the variational posterior $w \sim q(w|\theta)$.
2: Receive state $b$
3: Pick action $a$ with the highest expected reward according to the sampled parameters.
4: Receive reward $r$.
5: Update variational parameters $\theta$ as in Algorithm 4.
```

3.2.4 Deep BBQ-Learning

We implement the Bayes By Backprop method for DQN. For the objective function in Equ. 3.5, we use the expected square loss as in 3.2 for the likelihood term. Note that least-squares regression techniques can be interpreted as maximum likelihood with an underlying Gaussian error model.

The complete details of the algorithm are summarized in Algorithm 6.

Algorithm 6 BBQN

```
 1: Initialize replay memory $D$ to capacity $N$
 2: Initialize action-value function $Q$ with random weights $w_t$
 3: Initialize target action-value function $\hat{Q}$ with $w_t^- = w_t$
 4: for dialogue = 1, M do
 5:   Initialize $b_1$
 6:   for turn t = 1,T do
 7:     Sample weights from the variational posterior $w \sim q(w|\theta)$ using the reparamterization
        tick:
     8:       Sample $\epsilon \sim \mathcal{N}(0, I)$.
     9:       Let $w = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
10:     Receive state $b_t$
11:     Pick action $a_t$ with the highest expected reward according to the sampled parameters,
        $a_t = arg\max_a Q(b_t, a; \theta)$
12:     Take action $a_t$, observe $r_t$ and $b_{t+1}$
13:     Store transition $(b_t, a_t, r_t, b_{t+1})$ in $D$
14:     Sample random minibatch of transition $(b_j, a_j, r_j, b_{j+1})$ from D
15:     if Episode terminates at step j+1 then
16:       Set $y_j = r_j$
17:     else
18:       $y_j = r_j + \gamma max_{a'} \hat{Q}(b_{j+1}, a'; w_j^-)$
19:     end if
20:     Perform a gradient descent step on $f(w, \theta) = \xi_i(\log q(w|\theta) - \log P(w)) - \log P(\mathcal{D}|w)$ with
        respect to the network parameters $w$:
    21:       Calculate the gradient with respect to the mean $\mu$, and the standard deviation
        parameter $\rho$.
    22:       Update the variational parameters.
23:     Set $\hat{Q} = Q$ every C steps
24:   end for
25: end for
```

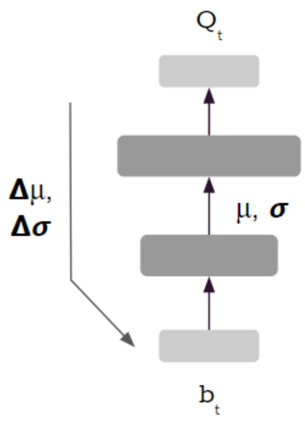


Figure 3.2: BBQN architecture using a feed-forward neural network. $\Delta\mu$ and $\Delta\sigma$ designate the gradients wrt the weight distributions parameters, Q_t is a Q scalar, b_t is a dialogue state.

3.2.5 α-Divergence

The approximate inference technique described in the Bayes By Backprop method corresponds to Variational Bayes (VB), which is a particular case of α-divergence, where $\alpha \to 0$ [18]. The α-divergence measures the similarity between two distributions and can take the form:

$$D_\alpha[p||q] = \frac{1}{\alpha(\alpha-1)}(1 - \int p(\theta)^\alpha q(\theta)^{1-\alpha} d\theta), \tag{3.11}$$

The following spacial cases have been explored for approximate inference by [18] and [6]:

$$D_1[p||q] = \lim_{\alpha\to1} D_\alpha[p||q] = \mathcal{KL}[p||q] \tag{3.12}$$

$$D_0[p||q] = \lim_{\alpha\to0} D_\alpha[p||q] = \mathcal{KL}[q||p] \tag{3.13}$$

$$D_{0.5}[p||q] = 2\int(\sqrt{p(\theta} - \sqrt{q(\theta)}^2)d\theta = 4Hel^2[p||q] \tag{3.14}$$

For $\alpha = 1$, expectation propagation (EP) [30] is recovered, and $alpha = 0.5$ corresponds to the Hellinger distance (Hel). Both [18] and [6] found that using $\alpha \neq 0$ performs better than the VB case, where an approximation with $\alpha \geq 1$ will cover all the modes of the true distribution, and the VB case only fits to a local mode, assuming the true posterior is multi-modal. $\alpha = 0.5$ achieves a balance between the two and has shown to perform best when applied to BNNs for regression or classification.

In attempt to improve the predictions of the Bayes By Backprop Q-Network (BBQN), we experiment with an objective function based on the black box α-divergence (BB- α) energy. We use the reparametrization proposed by [26] :

$$\mathcal{L}_\alpha \approx \check{\mathcal{L}}_\alpha = \mathcal{KL}[q(w|\theta)||P(w)] - \frac{1}{\alpha}\sum_n \log \mathop{\mathbb{E}}_{q(w|\theta)}[P(\mathcal{D}|w)], \tag{3.15}$$

where $\mathcal{L}_\alpha$ designates the BB- α energy, $\check{\mathcal{L}}_\alpha$ designates an approximation, and n corresponds to the number of datapoints in the minibatch. Using Monte Carlo sampling, the expression in 3.15

can be written as:

$$\mathcal{L}_{\alpha}^{MC}(q) = \mathcal{KL}[q(w|\theta)||P(w)] + const - \frac{1}{\alpha}\sum_{n} \log - sum - \exp[-\alpha \log P(\mathcal{D}|w)], \tag{3.16}$$

with the log-sum-exp operating over K samples from the approximate posterior $\hat{w}_k \sim q(w)$. This form has a similar form to the standard objective function in BNN. Note that for K=1, the log-sum-exp disappears, and the α's cancel out and the VB loss as in 3.5 is recovered. To explore the effect of α-divergences minimization, we sample the variational approximation $q(w)$ K times, with the standard VB loss as in BBQN, and compare the results with a model with the α-BB loss (α-BB-DQN), for the same number of K MC samples.

3.2.6 Dropout

Another technique to obtain uncertainty estimates in deep neural networks is Bayesian inference with dropout [10]. Dropout consists of randomly dropping units (along with their connections) from the neural network during training [42]. Dropout prevents overfitting and provides a way to train an ensemble of exponentially many different neural network architectures efficiently.

As explained in [26], dropout is equivalent to multiplying the NN weight matrices M_i by some random noise ϵ_i (with a new noise realization for each data point). The resulting stochastic weight matrices $W_i = \epsilon_i M_i$ can be seen as draws from the approximate posterior over the BNN weights.

The prediction uncertainty is obtained by marginalizing over the approximate posterior using Monte Carlo integration [26]:

$$\begin{aligned} p(y=c|x,X,Y) &= \int p(y=c|x,w)p(w|X,Y)dw && (3.17)\\ &\approx \int p(y=c|x,w)q_\theta(w)dw && (3.18)\\ &\approx \frac{1}{K}\sum_{k=1}^{K} p(y=c|x,\hat{w}_k) && (3.19)\end{aligned}$$

with $\hat{w}_k \sim q_\theta(w)$. For exploration, Thompson sampling is used instead of ϵ-greedy, which consists on performing a single stochastic forward pass through the network every time an action needs to be taken. Dropout is a simple technique as variational approximation, without the need to make major changes to the DQN implementation. Dropout variational inference is implemented by adding dropout layers before every weight layer in the neural network model. No changes to the objective function are necessary. We compare BBQN to DQN with Dropout (DQN-D).

3.2.7 Uncertainty in the loss function

In DQN, the neural network outputs a single value corresponding to the predicted Q-value, given a belief state input, and the parameters are optimized to minimize the root mean squared error (RMSE). Instead, we use a network that outputs two values in the final layer, as proposed in [33, 25], one that indicates the predicted mean $\mu(x)$, and the variance $\sigma^2(x) > 0$, indicating how certain the model is about the prediction. The likelihood captures how well a model fits the data, with larger values indicating better model fit. If one assumes the likelihood is a Gaussian distribution:

$$p_\theta(y_n|x_n) = \frac{1}{\sqrt{2\pi\sigma_\theta^2(x)}} \exp(-\frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)}). \tag{3.20}$$

Taking the log of both sides, we get:

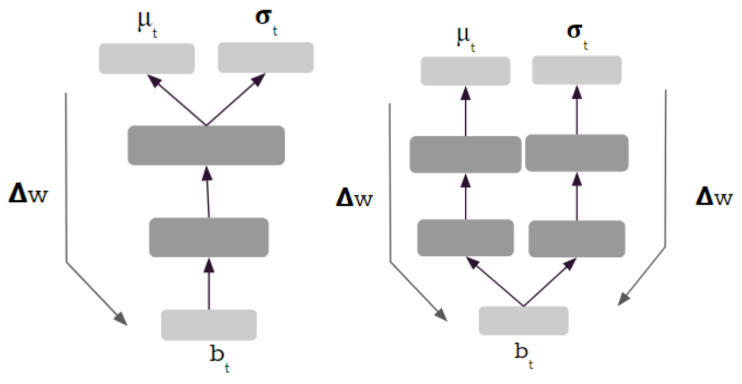
$$\log p_\theta(y_n|x_n) = -\frac{\log(2\pi)}{2} - \frac{\log(\sigma_\theta^2(x))}{2} - \frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)}. \tag{3.21}$$

The first term on the right is a constant and can be ignored for maximization. Since maximizing a value is the same as minimizing the negative of that value, parameters are optimized by minimizing the negative log-likelihood, which takes the following form:

$$-\log p_\theta(y_n|x_n) = \frac{\log(\sigma_\theta^2(x))}{2} + \frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)} \tag{3.22}$$

[25] found that this change to the training criterion performed better on a regression and classification task, then when optimizing for RMSE. Optimizing for the NLL coupled with Thompson sampling (DQN-NLL) is a simple method to extract uncertainty estimates from a NN, and we compare the performance of this method to the Bayes By Backprop method for DQN.

We attempt two different architectures for the neural network. Both output a mean and a variance for the Q values for the action, given a belief state input. One architecture shares the hidden layers for both outputs, whereas in the second architecture, the connections in the hidden layers are not shared, but both sets of hidden units are connected to the same input. Both architectures have the same objective function as in Eq. 3.22.



(a) (b)

Figure 3.3: DQN-NLL with (a) shared hidden layers configuration and with (b) two different networks connected to the same input.

3.3 Policy Based Learning

In addition to GP-SARSA, and DQN, the methods described above are compared to two policy based methods. We briefly explain how they work below.

3.3.1 Policy Based Methods

A policy-based method aims to find a parametrized policy $\pi_\theta(a|b) = \pi(a|b;\theta)$ that maximizes the expected reward over all possible dialogue trajectories given a stating state [43]. To approximate the policy, a policy gradient algorithm is used where the policy gradient is computed as [45]:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|b) Q^{\pi_\theta}(b,a)]. \tag{3.23}$$

Actor-critic methods use policy gradient to search in the policy space, and at the same time estimate a value function [15]. Two structures are used: an actor corresponding to the action-selection policy, and a critic corresponding to the value function. Their parameters are learnt simultaneously.

3.3.2 Advantage Actor-Critic (A2C)

In the advantage actor-critic (A2C) algorithm, Q^{π_θ} is parametrized separately with a weight vector (of a neural network), and the parameters are learnt as in DQN. The trained Q-network is the critic and is used for policy evaluation, while π_θ serves as the actor. Two separate deep neural networks are used: a Q-network and a policy network.

The form in Equation 3.23 cause high variance, and a baseline function is used instead, to reduce the variance without changing the gradient. The value function V(b) is used as the baseline as follow [43]:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|b) A_w(b,a)], \tag{3.24}$$

where $A_w(b,a) = Q(b,a) - V(b)$ is the advantage function. $A_w(b,a)$ is typically defined by two parameters sets θ and w. Temporal difference (TD) error $\varsigma_w = r_t + \gamma V_w(b_{t+1}) - V_w(b_t)$ is used to approximate the advantage function [43], reducing the number of parameters to estimate.

3.3.3 Episodic Natural Actor-Critic (eNAC)

Gradient descent algorithms like in A2C are not guaranteed to update the model parameters in the steepest direction due to reparametrization [43]. Instead, the advantage function can be approximated with $\nabla_w A_w(b,a) = \nabla_\theta \log \pi_\theta(a|b)$, where the update of w is in the same update direction as θ. This is the basis of the natural actor critic algorithm (NAC) where Equ. 3.24 is rewritten as:

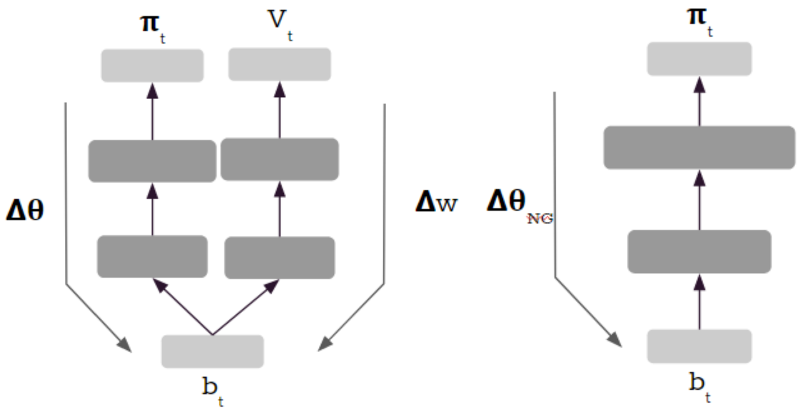
$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}[\nabla_\theta \log \pi_\theta(a|b) \nabla_\theta \log \pi_\theta(a|b)^T w] & (3.25)\\ &= F(\theta)w, & (3.26) \end{aligned}$$

where $F(\theta)$ is the Fisher information matrix, and $\Delta\theta_{NG} = w = F(\theta)^{-1}\nabla_\theta J(\theta)$ is the natural gradient and is independent of the policy parametrization. Directly using the natural gradient guarantees that parameters are updated in the direction of the steepest descent, which significantly speeds up convergence [43].

In the episodic version of the NAC algorithm (eNAC), the gradient is estimated by a least squares method given the n-th episode consisting of the tuples $\{b_t^n, a_t^n, r_t^n\}_{t=0}^{T_n - 1}$ [43]:

$$R^n = [\sum_{t=0}^{T_n-1} \nabla_\theta \log \pi_\theta(a_t^i|b_t^i)^T]\Delta\theta_{NG} + C, \tag{3.27}$$

where t designates the turn in the n-th episode, R^n is the discounted return for the n-th episode, and C is a constant and an estimate of the baseline $V(b)$. Equ. 3.27 can be solved analytically and the Fisher matrix does not need to be computed. eNAC is structured as a feed forward network with output π, and updated with natural gradient $\Delta\theta_{NG}$.



(a) (b)

Figure 3.4: (a) A2C and (b) eNAC architecture using a feed-forward neural network. $\Delta\theta$ is a policy gradient, Δw is a value gradient, $\Delta\theta_{NG}$ is a natural policy gradient, V_t is a value scalar, π_t is a policy vector, b_t is a dialogue state.

Chapter 4

Related Work

This work was in part motivated by the results obtained in [27]. Lipton et al [27] compared the performance of BBQN to DQN for policy optimization in a dialogue system for movie booking. They achieved a 0.48 final success rate with BBQN when the full domain was used, and 0.69 when smaller domain was used, then extended progressively during training. The agents were trained on 10k simulated dialogues. Experience replay and a target network were used in both BBQN and DQN. Their results outperformed DQN in both settings, indicating BBQN achieves more efficient exploration. They slightly improved final performance results over BBQN by using maximum a posterior (MAP) estimates to compute $y : y = r + \gamma_{a'} Q(b', a', \mu^{-})$. In addition, they implemented an exploration strategy based on maximizing the information gain about the agent's belief of environment dynamics, a methods called Variational Information Maximizing Exploration (VIME), introduced in [20]. This exploration method, which adds an intrinsic reward bonus to the reward function, only achieved slightly better results when the domain was extended progressively during training.

This thesis progresses the work in [27], and compares BBQN to other deep RL methods, and GPSARSA, which constitutes the state of the art for policy optimization for dialogue systems [13, 12]. In addition, we attempt methods other than Bayes By Backprop to get uncertainty estimates with DQN, and test the algorithms with different domains, and with different noise levels.

In [7], Fatemi et al compare deep RL methods (DQN, A2C) to GPSARSA and their results show that DQN outperforms GPSARSA in terms of convergence on summary spaces, however their results with GPSARSA are sensitive to the choice of kernel and the size of the dictionary used for sparcification. Further, they train GPSARSA with ϵ-softmax exploration and do not fully exploit the uncertainty that GPSARSA estimates. When evaluated with the original state and action spaces, DQN performs similarly to GPSARSA in terms of average reward and convergence, and outperforms GPSARSA in terms of wall-clock training time. The authors also show that two-stage training of the policy network in actor-critic based methods can exploit a small amount of data efficiently. In a two-stage setting, the policy network is trained with direct supervised learning, with few hundred dialogues collected with a handcrafted policy, and convergence to an optimal policy is significantly sped-up compared to other deep RL methods initialized on the data with batch RL. Their experiments are performed with simulated dialogues on the Cambridge Restaurants domain.

More recent work on dialogue management has focused on sample efficient actor-critic based algorithms [43]. Su et al. apply eNAC with experience replay and an off-policy trust region actor-critic with experience replay (TRACER) to policy optimization. With eNAC and TRACER, they improve sample-efficiency and achieve better performance at the early stages of the training. For TACER, the trust region incorporates a KL divergence constraint when updating the policy. This controls the learning step size and avoids radical changes to the policy. They further improve performance by pretraining their eNAC and TRACER models with a corpus of demonstration data, prior to on-line RL.

This thesis touches on the subject of Bayesian neural networks and deep reinforcement learning as well. Efficient exploration in RL is an active area of research, and a recent paper by Fortunato et al [8] introduces NoisyNets, a deep reinforcement learning agent with parametric noise added to its weights.The weights and biases in the neural networks are perturbed by a parametric function of the noise and the parameters are adapted with gradient descent. When NoisyNets were adapted to DQN and A2C, results show significant performance improvements across many Atari games, when compared to regular DQN, and A2C. Their experiments do not compare however NoisyNets to the Bayes By Backprop method

Another method to extract uncertainty estimates from DNNs is the bootstrapped NN method by Osband et al [35]. The authors show that the bootstrap with random initialization can produce reasonable uncertainty for neural networks at low computational cost. The method is parallelizable and consists on generating bootstrapped samples from a network with K bootstrapped "heads" branching independently from a shared architecture. Each head is trained on its bootstrapped sub-sample of the data. They apply this method to DQN and show that bootstrapped DQN leans much faster than DQN, and improves upon the final score across most experiments with Atari games.

Both the bootstrapped DQN and the NoisyNets method could be interesting future avenues to explore for deep RL for policy optimization.

Chapter 5

Experiments

The CUED software toolkit PyDial [47] was used to conduct the experiments. The toolkit provides a platform for modular spoken dialogue system. The toolkit includes an agenda-based user simulator, a focus tacker, the ontology for the domains and the policy optimization modules.

5.1 User Simulator

We use a user simulator to train the models. While direct interaction with users is the most straightforward way to optimize a POMDP-based dialogue system, a sufficient number of real users is usually not available. A user simulator replicates user behavior with sufficient accuracy to optimize model parameters to an acceptable level of performance [51], and is more cost-effective for development and evaluation purposes.

The user simulator in the PyDial toolkit is agenda-based [39] and generates user interactions at the semantic level. The goal and the agenda are updated dynamically throughout the dialogue, and updates are controlled by decision points that can be deterministic or stochastic, enabling a wide spread of realistic dialogues to be generated [12].

We use an error model where confusions to the simulated user input are added. The error model outputs an N-best list of possible user responses. We present results with experiments where no noise is added to the simulated user input, and with experiments with different noise levels. For experiments with added noise levels, a confusion rate of 15% is used during training (i.e. 15% of the time, the true hypothesis is not in the N-best list), and the models are evaluated on simulated dialogues with 0%, 15%, 30%, and 45% confusion rates.

The reward function is set to give a reward of 20 for successful dialogues, zero for unsuccessful dialogues, and 1 is deducted for each turn in the dialogue. The discount factor γ is set to 0.99 and the dialogue length is limited to 25 turns.

5.2 Domains

Experiments are conducted using the Cambridge restaurant domain and the Cambridge Hotel domain from the PyDial toolkit, and only the summary action state space is used. The Cambridge restaurant domain consists of a selection of about 150 restaurants, with 8 slots for every restaurant. 4 slots are used to constrain the search (food-type, area, price-range, restaurant type) and 4 are system-informable properties (phone-number, address, postcode, name). The input for all models is the full dialogue belief state b of size 268, which includes the last system act and distributions over the user intention and the requestable slots. The action space consists of 14 actions. The Cambridge hotel domain consists of a selection of 40 hotels, with 13 slots, where 3 are informable (address, name, phone) and 8 are used for constraints (price, price range, stars, kind, type, takes booking, has internet, has parking). The input is of size 111, and the action space consists of 21 actions.

5.3 Training Details

All models are trained over 4000 simulated dialogues. The experience replay pool size is 1000, and minibatches of 64 are used. Once 192 samples are collected, the model is updated every 2 dialogues. For DQN, BBQN, DQN-NLL, α-BB-DQN, and D-DQN, each sample is a state transition (b_t, a_t, r_t, b_{t+1}), whereas in A2C, and eNAC, each sample includes the whole dialogue with all its transitions.

All deep RL models (BBQN, DQN-NLL, D-DQN, α-BB-DQN, DQN, eNAC, A2C) contain two hidden layers of size 130 and 50. The Adam optimiser [24] is used with an initial learning rate of 0.001. For A2C, eNAC and DQN, an ε-greedy policy is used, which is initially set to a value, and annealed to 0.0 after 4000 training dialogues. The initial value was found by grid search over the Cambridge Restaurants domain with 0% confusion rate, and best results were obtained with $\varepsilon_{initial} = 0.5$ for DQN, eNAC and A2C.

For the Bayes By Backprop method, the mean μ of the weights is initialized with $\mathcal{N}(0, \sigma_\mu)$ and the parameter ρ, with $\mathcal{N}(\mu_\rho, \sigma_\rho)$. For the initialization parameters σ_μ, μ_ρ and σ_ρ, a grid search over the range $\{1.0, 0, 5, 0.1, 0.05, 10^{-2}, 10^{-3}, 10^{-4}, 0.0\}$ is performed. All value ranges for the grid searches for tuning hyperparameters for BBQN were chosen based on previous experiments with BBQN for classification and regression tasks [2], and based on the work by Lipton et al [27].

For the variational inference with dropout (DQN-D), we perform a grid search over dropout probabilities of $\{0.5, 0.35, 0.25, 0.15\}$, applied to the first layer only, the second layer only or both layers at the same time, during training only, at testing only, or both during training and testing. Experiments are also run with a dropout probability that decreases as training progresses, with a starting dropout probability of $\{0.5, 0.65, 0.75, 0.85, 1.0\}$.

Parameter tuning is only done on the Cambridge restaurant domain with 0% confusion rate. The parameters with best performance rate are then used for training and testing on the Cambridge Hotel domain, and on the Cambridge restaurant domain with different levels of confusion.

This thesis involved implementing the algorithms for BBQN, D-DQN, α-BB-DQN, and DQN-NLL. For the other models presented here, the current implementations in the PyDial toolkit are used.

5.4 Evaluation

3 to 5 training sessions are seeded with different random seeds, and after every 200 training dialogues, the performance of the partially trained policies is evaluated with 400 dialogues, ensuring the simulated user has a different random seed than the one used in training. The results are averaged over the evaluation dialogues. This evaluation schedule allows to examine the variability in performance and avoids relying on a good performance due to a lucky choice of actions at the beginning of training.

Chapter 6

Results

6.1 Tuning BBQN Hyperparameters

6.1.1 Setting σ_{prior}

Th BBQN model requires to determine the variance of the Gaussian prior distribution over the weights $P(w)$, which is set to be isotropic. A grid search over the values $\exp(\{-3, -1.5, 0, 1.5, 3, 4.5, 6\})$ is performed to find σ_{prior} with best performance. The average success rate for the different settings of σ_{prior} is shown in Fig 6.1. In general, a smaller σ_{prior} constrains the posterior distribution over the weights to have smaller variance, which can result in under-exploration and convergence to a suboptimal policy as seen with $\sigma_{prior} = \exp(-3.0)$. On the other hand, larger σ_{prior} can lead to over-exploration as seen with $\sigma_{prior} = \exp(6.0)$. The setting of σ_{prior} that performs best is found to be between $\exp(1.5)$ and $\exp(3.0)$, and σ_{prior} is fixed at 1.5.

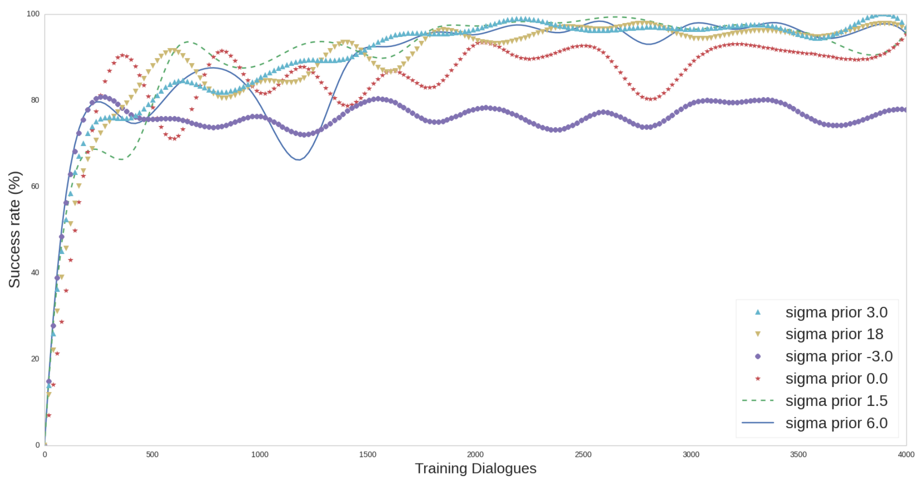


Figure 6.1: Influence of σ_{prior} on the performance of BBQN. Average success rate over one run for each setting of σ_{prior} is shown. The Cambridge Restaurant domain is used.

6.1.2 Weighting the $\mathcal{KL}$ divergence

Another hyperparrameter to tune is the weight of the $\mathcal{KL}$ divergence term in the objective function for BBQN. Recall:

$$\mathcal{F}_i(\mathcal{D}_i, \theta) = \xi_i \mathcal{KL}[q(w|\theta)||P(w)] - \mathop{\mathbb{E}}_{q(w|\theta)}[\ln P(\mathcal{D}_i|w)], \tag{6.1}$$

where ξ_i is the weight for the $\mathcal{KL}$ cost, for minibatch i. There are many ways to weight the complexity cost relative to the likelihood cost on each minibatch. One approach suggested by [16] is to weight the $\mathcal{KL}$ term with $\frac{1}{M}$, where M is the total number of minibatches used during training. Another approach is to use the scheme $\xi_i = \frac{2^{M-i}}{2^M-1}$ [2], where the first few minibatches are heavily influenced by the complexity cost, and as more data are seen, data become more influential and the prior less influential.

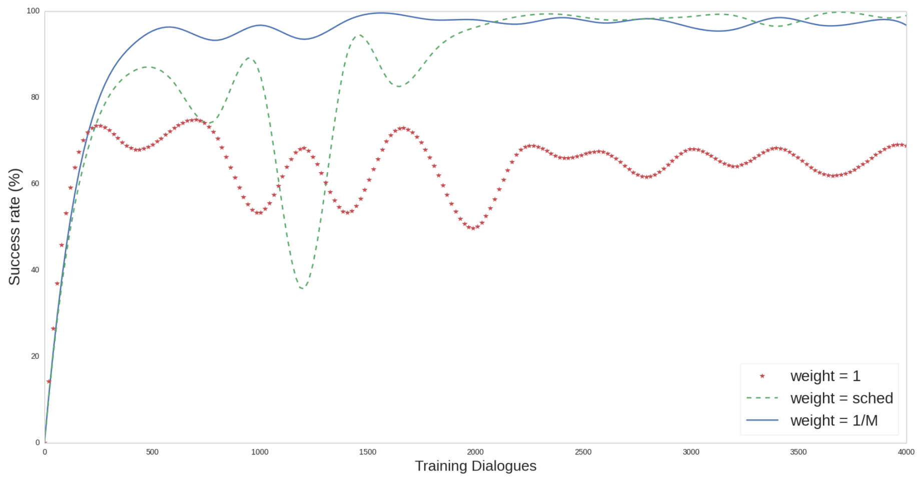


Figure 6.2: Influence of different schemes for weighing the $\mathcal{KL}$-divergence term in the objective function on the performance of BBQN. Average success rate over one run for each scheme is shown. `sched` designates the scheme with $\xi_i = \frac{2^{M-i}}{2^M - 1}$, and M designates the number of mini-batches used for training. The Cambridge Restaurant domain is used.

Fig 6.2 compares success rate performance for different weight schedules for the $\mathcal{KL}$ divergence term. The best performance is achieved when ξ_i is set to $\frac{1}{M}$. A schedule that reduces ξ_i after each update does not lead to better performance, mainly because as the training progresses, the mean squared error starts to dominate the objective function and the variational distribution moves further from the prior. Further scaling down the $\mathcal{KL}$ term as training progresses is unnecessary, and leads to unstable parameter updates (Figures A.1 - A.6 in appendix A illustrate how the loglikelihood and the $\mathcal{KL}$ term change as training progresses, for different settings of ξ_i). When the $\mathcal{KL}$ term is not scaled down, it dominates the objective function (see Fig. A.3 in appendix A, the $\mathcal{KL}$ is about 1000 larger than the loglikelihood term), as a result the model does not learn, and achieves premature convergence to a poor solution.

6.1.3 Varying σ_ϵ

In order to obtain an unbiased estimator of the gradient and to be able to back-propagate through the layers that sample w from $q(w|\theta)$, the reparamatization tick is used. This trick consists of sampling $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$, then computing $w_i = \mu_i + \sigma_i \circ \epsilon_i$. We perform a grid search to determine an appropriate value of σ_ϵ. Results are shown in Fig 6.3.

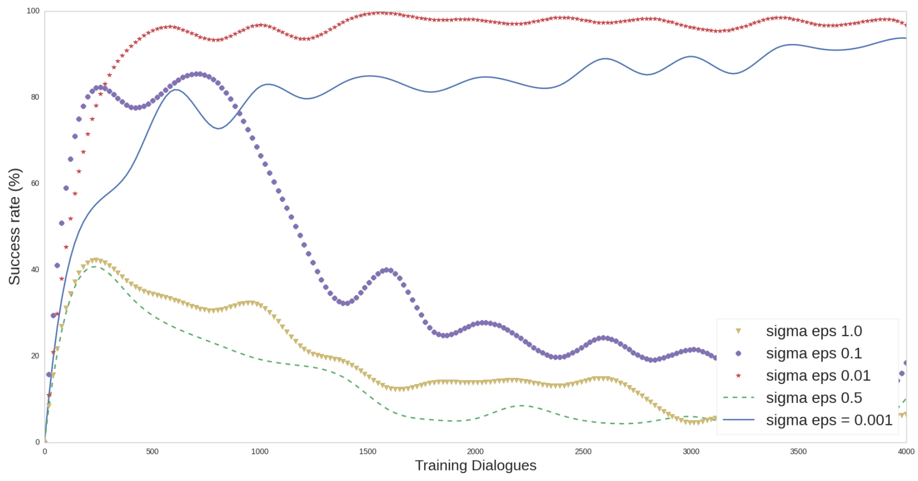


Figure 6.3: Influence of σ_ϵ on the performance of BBQN. Average success rate over one run for each setting of σ_ϵ is shown. The Cambridge Restaurant domain is used.

Larger values of σ_ϵ tend to result in gradient estimators with higher variance, which diverge from an optimal solution, while smaller values of σ_ϵ result in much slower convergence to an optimal solution. We find a value of $\sigma_\epsilon = 0.01$ leads to an optimal policy with fast convergence.

6.2 Comparison with baselines

6.2.1 The Cambridge Restaurants Domain

In Figures 6.4 and 6.5, we show the average success rate, and the average reward for BBQN, GPSARSA, DQN, A2C and eNAC, for test results for the Cambridge Restaurants domain, with a noise-free environment. Results are averaged over 5 runs with different random seeds.

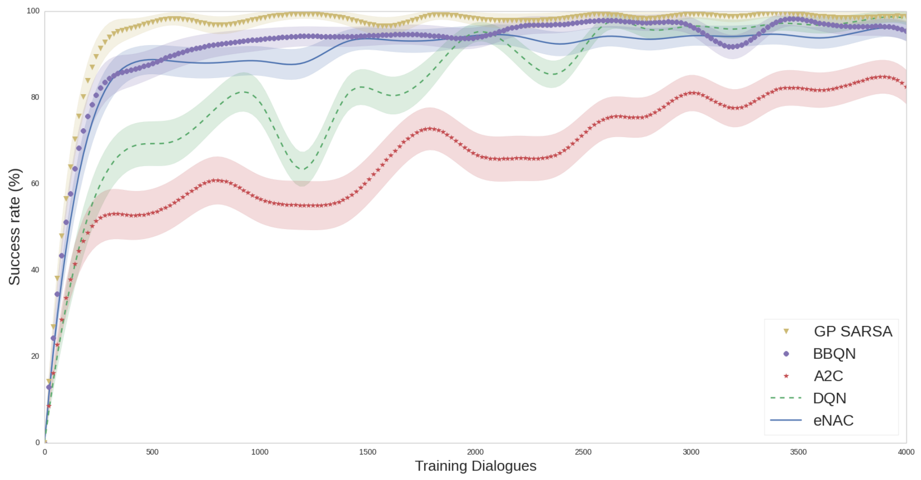


Figure 6.4: The success rate learning curves for BBQN, DQN, A2C, eNAC and GPSARSA for the Cambridge Restaurant domain, under noise-free conditions, with two standard error bars. Results are averaged over 5 runs.

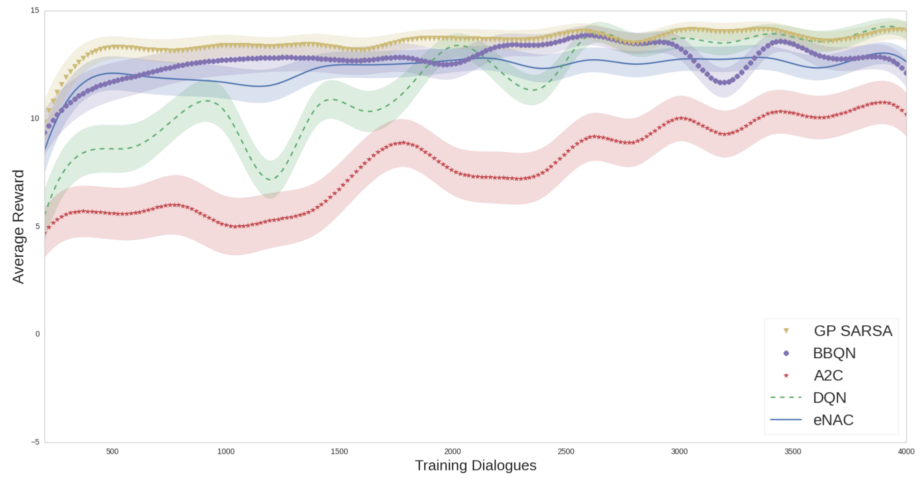


Figure 6.5: The average reward as a function of the training dialogues for BBQN, DQN, A2C, eNAC and GPSARSA for the Cambridge Restaurant domain, under noise-free conditions, with two standard error bars. Results are averaged over 5 runs.

Model	**Final Performance Success Rate**	**Final Performance Average Reward**
eNAC	95.3	12.632
DQN	98.3	14.074
A2C	82.45	10.19
BBQN	95.4	12.116
GPSARSA	98.75	14.06

Table 6.1: Final performance of trained models on 4k simulated dialogues, averaged over 5 runs.

When evaluated on the Cambridge Restaurants domain, GPSARSA learns the fastest and is the most stable, benefiting from the ability of Gaussian Processes to learn from a small amount of data, exploiting the correlations defined by the kernel function. The results show that BBQN and eNAC reach a performance comparable to GPSARSA, and outperform A2C, and DQN in general. BBQN learns slightly faster than eNAC. DQN reaches a higher final success rate than BBQN and a more stable performance at final stages of the training, but converges much slower, with high instability and more exploration than needed at some points.

6.2.2 The Cambridge Hotel Domain

To confirm the observations in section 6.2.1, we evaluate the same models on the Cambridge Hotel Domain. This domain includes more constraint slots and is more complex than the Cambridge Restaurants domain. Parameters were not tuned on this domain, and the ones that led to best performance on the Cambridge Restaurant domain are used for the Hotel domain. Average Success Rate and Average Reward are shown in Figures 6.6 and 6.7. Results are averaged over 3 runs with different random seeds.

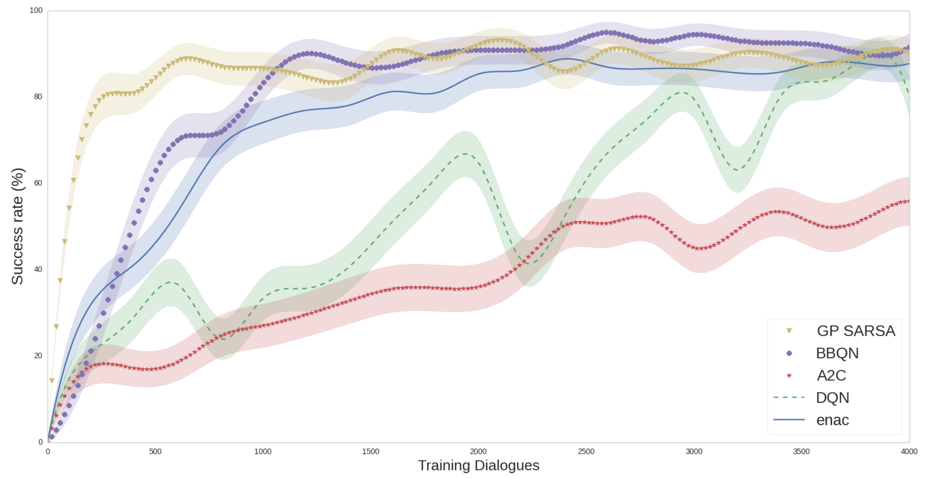


Figure 6.6: The success rate learning curves for BBQN, DQN, A2C, eNAC and GPSARSA for the Cambridge Hotel domain, under noise-free conditions, with two standard error bars. Results are averaged over 3 runs.

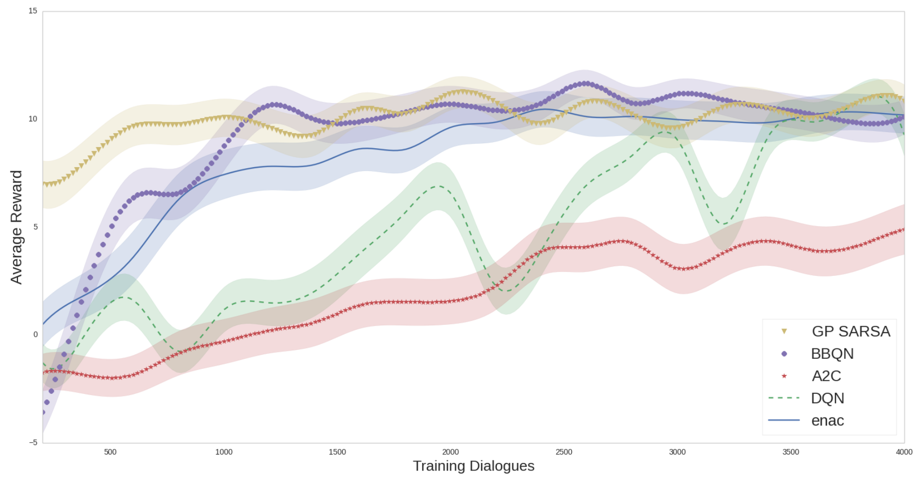
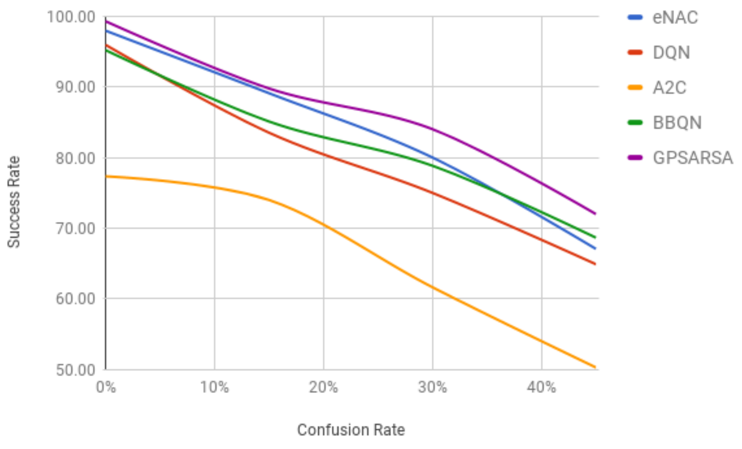


Figure 6.7: The average reward as a function of the training dialogues for BBQN, DQN, A2C, eNAC and GPSARSA for the Cambridge Hotel domain, under noise-free conditions, with two standard error bars. Results are averaged over 3 runs.

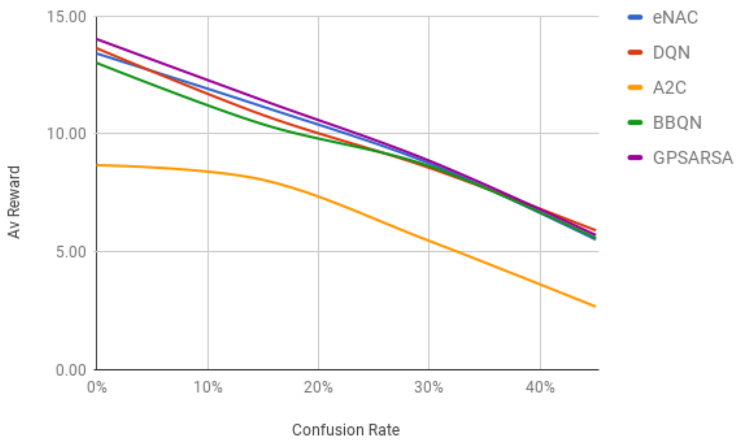
The results on the Cambridge Hotels domain are consistent with the ones observed on the Cambridge Restaurants domain, with BBQN outperforming DQN, eNAC, A2C in terms of convergence and overall success rate and average reward. BBQN is much slower in terms of convergence to an optimal policy than GPSARSA but achieves slightly better success rate and average reward than GPSARSA at certain points, especially in later stages of training. This further confirms the capability of BBQN to perform as well as GPs on policy optimization, especially when tested on more complex domains.

6.2.3 Performance across different confusion rates

The policy is trained on the simulated user by performing 4000 dialogues with a 15% confusion rate, then evaluated over the range of confusion rates $\{0, 15, 30, 45\}\%$. The final success rate and final average reward for each of the 5 models, averaged over 3 runs, are given in Fig. 6.8 as a function of the confusion rate at testing. Success rate as a function of the training dialogues, for all models, is shown in Fig. 6.9.

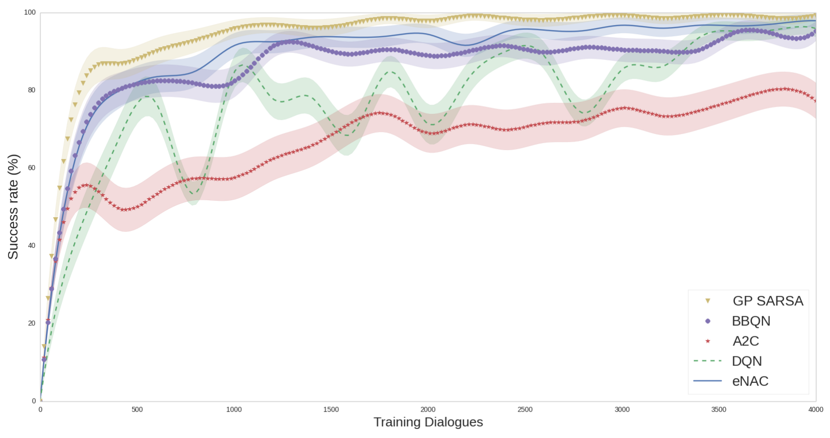


(a)

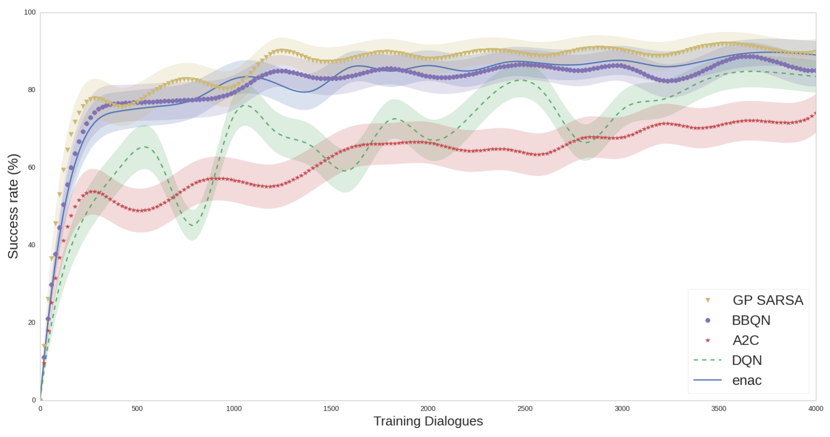


(b)

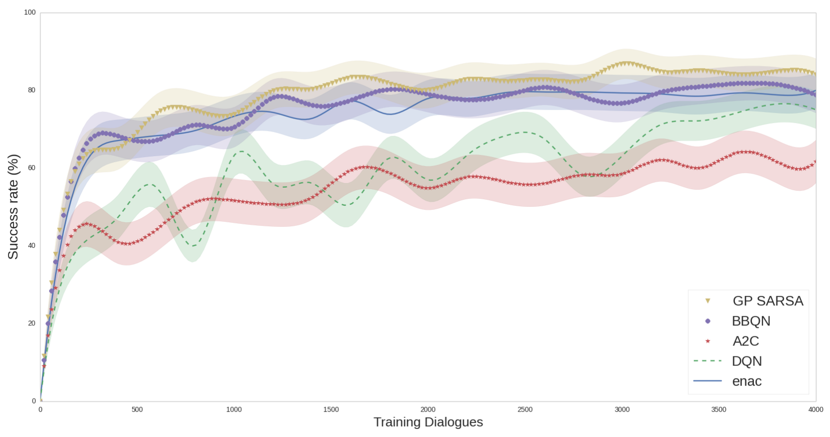
Figure 6.8: (a) Final success rate and (b) final average reward (after training for 4000 dialogues) as a function of the confusion rate at testing, for BBQN, DQN, A2C, eNAC and GPSARSA. Results are shown for the Cambridge Restaurant domain, and are averaged over 3 runs. The confusion rate at training is 15%.


100
80
60
40
20
0
Success rate (%)
0
500
1000
1500
2000
2500
3000
3500
4000
Training Dialogues
GP SARSA
BBQN
A2C
DQN
eNAC

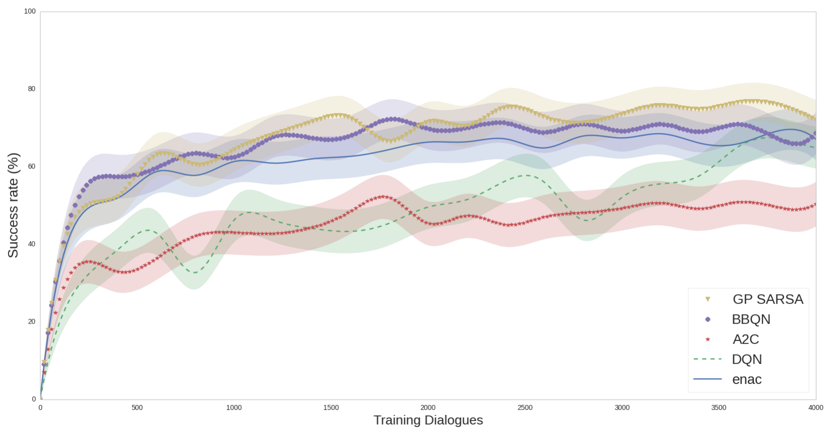
(a) 0%


100
80
60
40
20
0
Success rate (%)
0
500
1000
1500
2000
2500
3000
3500
4000
Training Dialogues
GP SARSA
BBQN
A2C
DQN
enac

(b) 15%



(c) 30%



(d) 45%

Figure 6.9: Success rate for BBQN, DQN, A2C, eNAC, and GPSARSA with a (a) 0%, (b) 15%, (c) 30% and (d) 45% confusion rate at testing, and 15% confusion rate during training. Results are shown for the Cambridge Restaurant domain with two standard error bars and are averaged over 3 runs.

The results show that GP SARSA performs best in terms of success rate, followed by BBQN, across all confusion rates. As the confusion rate increases at testing, BBQN performs almost as good as GPSARSA. We also note that the performance of BBQN does not decrease as fast as any other algorithm, especially at higher confusion rates at testing. The final average reward when the confusion rate is 45% at testing is highest for BBQN. This shows that BBQN generalizes better than ε-greedy algorithms. BBQN has the potential for robust performance, and performs well, even at conditions different from the training conditions.

6.2.4 Discussion

BBQN achieved much faster learning than ε-greedy DQN. This is mainly because using uncertainty estimates allows for more efficient exploration, and ε-greedy methods do not ensure policy improvement when applied with an approximate value function [36]. During a greedy update, small errors in the value function can cause large changes in the policy which in return can cause large changes in the value function. This process, when applied repeatedly, can result in oscillations or divergence of the algorithms.

Further, BBQN performs better than other deep RL methods and even GPSARSA when used on more complex domains. The generalization capabilities of BBQN are also higher than all baselines considered in this thesis. The final performance for BBQN decreases the least as the confusion rate at testing increases.

Policy gradients methods have strong convergence guarantees, even when used in conjunction with approximate value functions. However, large plateaus in the expected return landscape where the gradients are small and where the gradient often do not point directly towards the optimal solution [36] can result in very low performance, and in inefficient learning, as shown with the results obtained with A2C. By using natural gradients computed using the Fischer matrix, convergence to the next local optimum can be assured with eNAC, resulting in much more efficient learning than A2C, and a performance similar to BBQN and GPSARSA. Using episodic NAC guarantees the unbiasedness of the natural gradient estimate [36].

While BBQN reaches comparable performance to GPSARSA and achieves much more efficient exploration than DQN, performance to which BBQN converges is not as good as with GPs. This is because variational inference as in the Bayes By Backprop method tends to underestimate model uncertainty, resulting in slightly under-exploration and convergence to a sub-optimal policy. Minimizing the $\mathcal{KL}$ divergence between $q(w|\theta)$ and $p(w|\mathcal{D})$ penalizes $q(w|\theta)$ for placing probability mass where $p(w|\mathcal{D})$ has no mass, but does not penalize $q(w|\theta)$ for not placing probability mass at locations where $p(w|\mathcal{D})$ does have mass [10].

6.3 Limitations of BBQN

As explained earlier, the variational method in BBQN can underestimate uncertainty which could lead to under-exploration and convergence to a sub-optimal policy. In addition to the following limitation of BBQN, there are few others setbacks that come when using Bayes By Backpropagation.

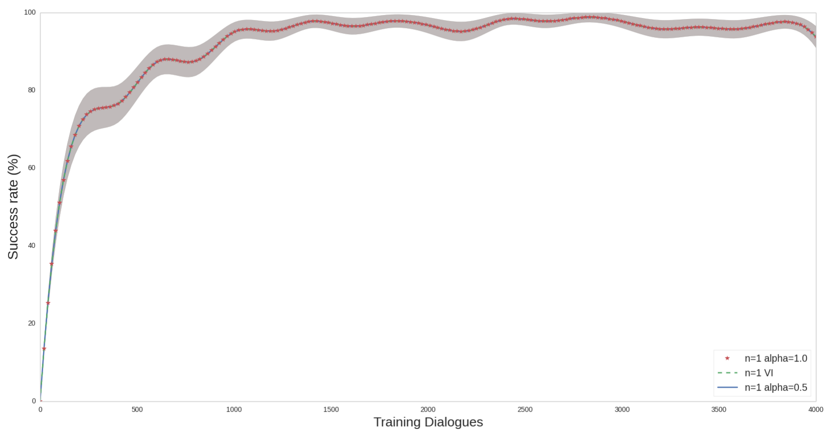
The predictive distribution obtained with variational Bayes depends on the degree of approximation (due to computational constraints) and the choice of the prior distribution. The assumptions about the shape of the distribution over the weight (a Gaussian with diagonal covariance) may not hold and may lead to poorer performance.

In addition, Bayesian neural networks require doubling the number of computations which make them slower to train and harder to implement, compared to non Bayesian neural networks. Further, the number of hyperparameters to tune is much higher than non Bayesian neural networks or GPs, and a grid search was necessary to complete the work in this thesis. This grid search was computationally expensive and long, and was restricted to few values for each hyperparameter.

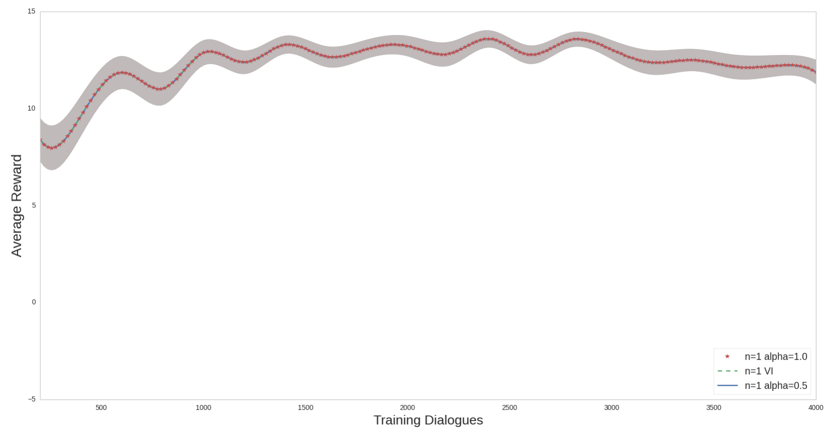
Nonetheless, BBQN achieves more efficient exploration than DQN, and compared to GPs, BBQN is less computationally complex. Memory requirements can also be determined in advance, unlike GPSARSA.

6.4 α-Divergences

As explained in the previous section, $\mathcal{KL}$-divergence enforces the q distribution to be zero in the region where the exact posterior has zero probability mass. VI often fits to a local mode of the exact posterior and is over confident in prediction [18]. This problem can be addressed with α-divergences. For instance, α-divergence with $\alpha = 1$ encourages the coverage of the support set and can fit the approximation to a mode of p with better estimates of uncertainty. Hellinger distance ($\alpha = 0.5$) provides a good balance between zero-forcing and mass-covering. Figures 6.10, 6.11 and 6.12 show results for different settings of α and K, the number of stochastic forward passes during training. Results are for one run, on the Cambridge Restaurants domain.

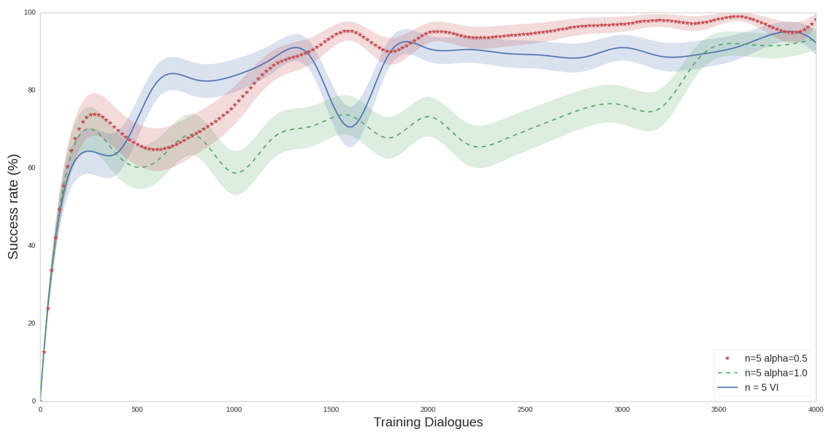


(a)

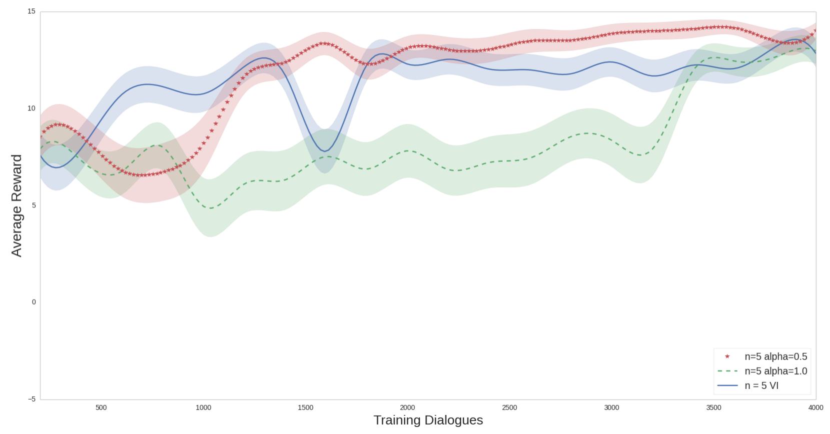


(b)

Figure 6.10: (a) Success rate and (b) average reward for BBQN with variational inference (VI) corresponding to $\alpha = 0.0$-divergence, and for $\alpha = 0.5$ and $\alpha = 1.0$ divergences, for $K = 1$ MC samples. Results are shown for the Cambridge Restaurant domain, with two standard error bars, with noise-free environment, and are averaged over 1 run.

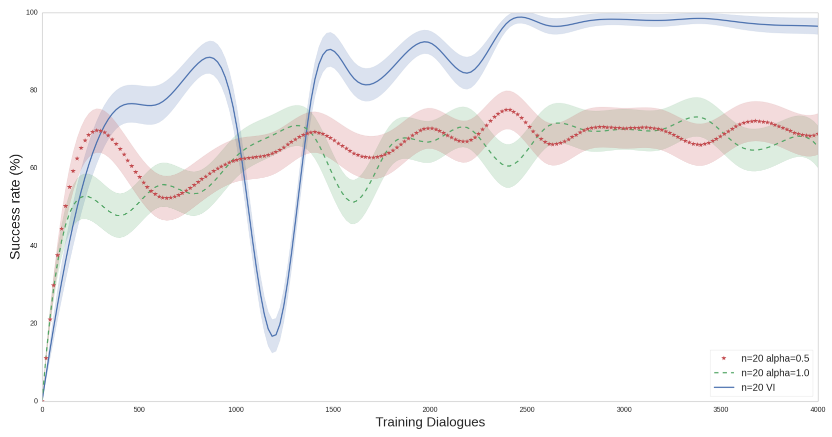


(a)

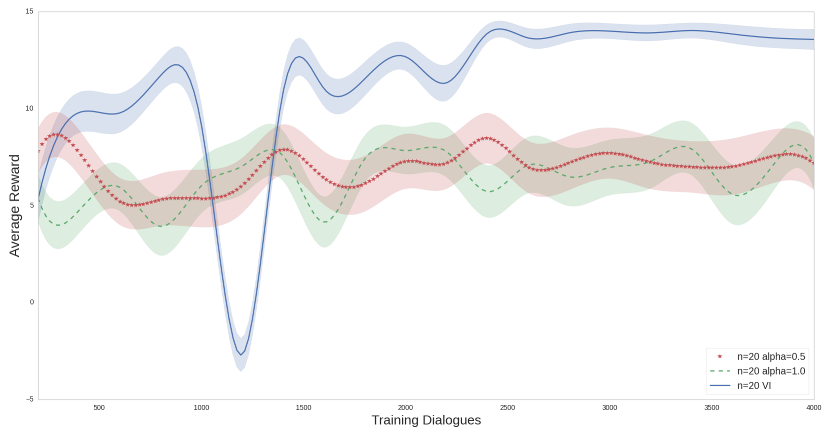


(b)

Figure 6.11: (a) Success rate and (b) average reward for BBQN with variational inference (VI) corresponding to $\alpha = 0.0$-divergence, and for $\alpha = 0.5$ and $\alpha = 1.0$ divergences, for $K = 5$ MC samples. Results are shown for the Cambridge Restaurant domain, with two standard error bars, with noise-free environment, and are averaged over 1 run.



(a)



(b)

Figure 6.12: (a) Success rate and (b) average reward for BBQN with variational inference (VI) corresponding to $\alpha = 0.0$-divergence, and for $\alpha = 0.5$ and $\alpha = 1.0$ divergences, for $K = 20$ MC sample. Results are shown for the Cambridge Restaurant domain, with two standard error bars, with noise-free environment, and are averaged over 1 run.

The Hellinger value $\alpha = 0.5$ or other settings of α did not perform better than VI in geneal, and for $K = 20$ samples at training, α-divergences did not achieve any learning. We also note, that convergence to an optimal policy is slower with increasing number of samples. For instance, settings with $K = 5$ samples reach a success rate of 90% after 1600 training dialogues, whereas settings with 1 sample reach the same success rate after 1000 training dialogues. The final success rate with $K = 20$ samples is much lower for all α than trainings with 1 MC sample. Taking more MC samples decreases the variance of the gradient estimates, and the averaged loss for most updates is closer to the loss obtained when taking a sample close to the mean of the variational distribution q. This implies more updates are necessary to move in the direction of the true posterior distribution p (by minimizing the $\mathcal{KL}$ or α divergence, trading off for reduced exploration, and slower learning). With the BB-α reparametrization, more than 1 samples are necessary for α-divergence to have any effect, but taking more than 1 sample at training reduces exploration efficiency and results in worse performance.

6.5 Dropout

The best performance obtained with variational inference with dropout, is the model with dropout with probability 0.25 applied before every hidden weight layer. Figures 6.13 and 6.14 shows the performance of this model (DQN-Dropout) compared to BBQN and DQN. We also show the performance of a model with an annealed dropout rate with a starting rate of 0.85. Note that a grid search over different starting dropout probabilities, with dropout applied to one layer only, or both layers was also performed. Only the best performance over this grid search is shown.

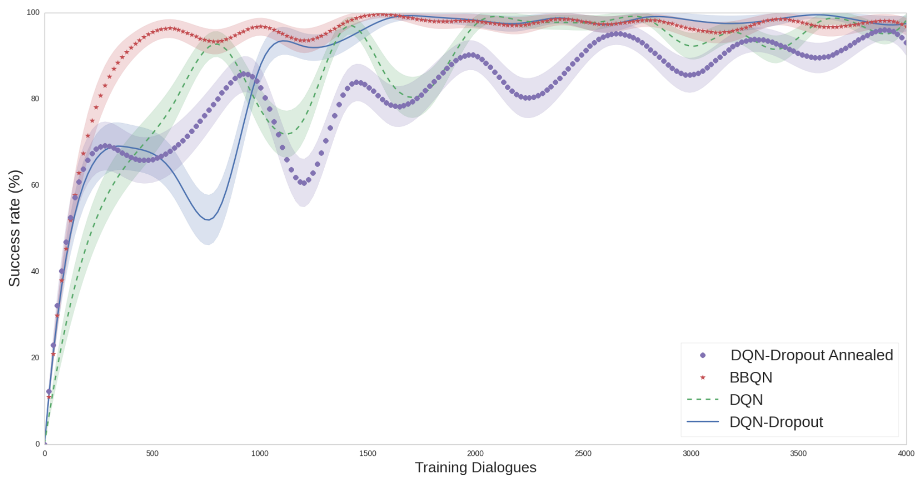


Figure 6.13: Success rate performance of DQN with variational dropout (DQN-Dropout), compared to BBQN and DQN. `DQN-Dropout Annealed` designates a dropout rate that decreases as training progresses. `DQN-Dropout` designates a fixed dropout rate of 25%. Results are shown for the Cambridge Restaurant domain, with two standard error bars, with noise-free environment, and are averaged over 1 run.

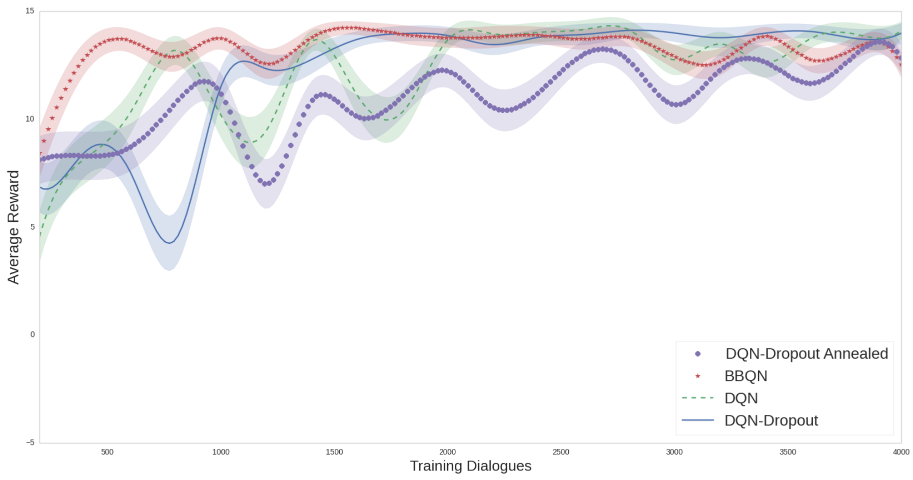


Figure 6.14: Average reward of DQN with variational dropout (DQN-Dropout), compared to BBQN and DQN. `DQN-Dropout Annealed` designates a dropout rate that decreases as training progresses. `DQN-Dropout` designates a fixed dropout rate of 25%. Results are shown for the Cambridge Restaurant domain, with noise-free environment, and are averaged over 1 run.

The plot shows the model with a fixed dropout rate converges much faster than DQN and achieves a performance similar to BBQN after 1000 training dialogues. Dropout inference is also more stable than DQN after convergence, but has lower performance at the early stages of training. While deceasing the dropout probability as the amount of data increases and as the model explains away its uncertainty may suggest a better performance, such schedule performs worse than DQN in the experiments conducted here.

Using dropout is a very simple method to perform variational inference, achieves faster convergence and more stable leaning that ε-greedy DQN, and does not require doubling the number of parameters as in BBQN. However, the model with dropout does not achieve the same learning observed with BBQN, and a grid-search over the dropout probabilities is required. Such grid search can result in a large waste of computational resources, and prolonged experimentation cycles. For the work in this thesis, the grid search was constrained to a small number of possible dropout values to make the search feasible, which might have hurt the uncertainty calibration cycles. In [11], Gal et al suggest a method for automatic tuning of the dropout. This method consists of a continuous relaxation of dropout's discrete masks, and an optimization of the dropout probability using gradient methods. Future work can include an attempt to use such method to find better dropout values for DQN for policy optimization.

6.6 DQN-NLL

An alternative to the Bayes By Backprop method to get uncertainty estimates in neural networks is to minimize the negative log-likelihood criterion. The variance of the likelihood is an output in the final layer and determines the uncertainty estimate about the Q value prediction, which is determined by a mean value, also an output of the final layer. We implement this method (DQN-NLL) and compare the performance to BBQN to DQN.

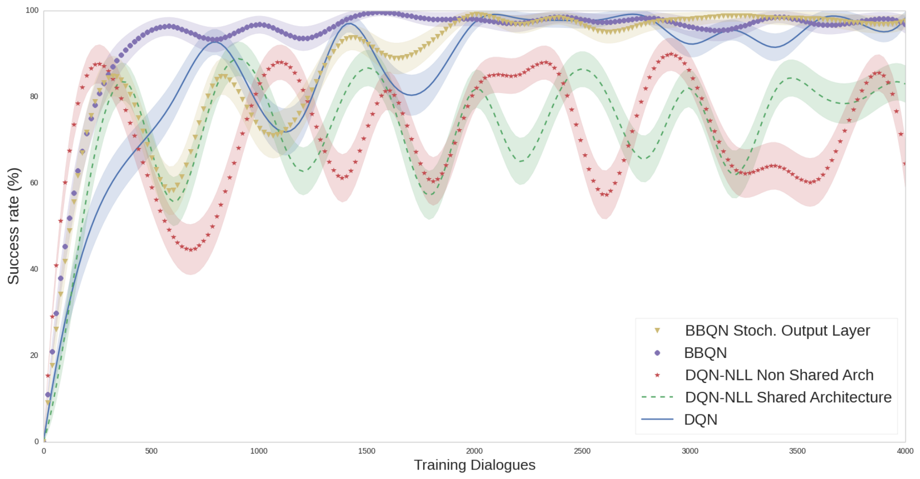


Figure 6.15: Success rate performance of DQN-NLL, compared to BBQN and DQN. `DQN-NLL Shared Architecture` designates the version with a shared architecture for the hidden layers except for the outputs layers. `DQN-NLL Non Shared Arch` designates the version where two networks are used, one for each of the two outputs μ and σ. `BBQN Stoch Output Layer` designates a version of BBQN where only the output layer is stochastic, and where the rest of the hidden layers is deterministic (fixed points estimates for the weights instead of a distribution over weights). Results are shown for the Cambridge Restaurant domain, with two standard error bars, with noise-free environment, and are averaged over 1 run.

Fig 6.15 shows the performance of DQN-NLL. Minimizing the negative loglikelihood and using the variance to get predictive uncertainty does not improve on the performance of BBQN and DQN. We spent some time altering the scheme to fix the poor performance obtained with both architectures, which might be due to an artefact in the implementation. The only change that led to better performance is the one where the weights for the output layer of the NN were changed to be stochastic. A MC sample is then taken, and instead of using this sample as the output (as expected in DQN-NLL), it was used to obtain a sample of the weights and perform a forward pass as in BBQN. This change is only a simple version of BBQN and does not correspond to the method described in section 3.2.7. Further, this change only slightly outperforms DQN, in that the performance is more stable after convergence to a solution.

One reason DQN-NLL does not perform as expected might be that the Gaussian distribution might be too restrictive. This can be addressed by using a mixture density network or a heavy tailed distribution instead [25]. It is also possible to impose a prior and perform maximum-a-posteriori (MAP) estimation, in attempt to get better results.

In addition to the method outlined here, Lakshminarayanan et al [25] suggest the use of ensembles and adversarial training to obtain well-calibrated uncertainty estimates. The ensembles are implemented by training randomly initialized neural networks, along with random shuffling of data points. These methods could be potentially investigated in future work.

6.7 Future Work

Future work needs to address uncertainty estimates in deep neural networks to increase their sample efficiency when used for policy optimization. As pointed out in previous sections, there are many avenues to explore. For instance, noisy networks with parametric noise added to the weights can be used, and the authors in [8] have applied this method to deep RL specifically.

Results obtained with variational dropout were promising, and improvements could be obtained with a better tuning of the dropout probability, which can be automatically done by continuously relaxing the dropout's discrete masks and by optimizing the dropout with gradient methods, as suggested in [11]. Another work of interest regarding dropout is the recent paper by Li and Gal [26] where they demonstrate that variational dropout with α-divergences improve uncertainty estimates and accuracy on regression and classification tasks, compared to VI in dropout networks. This method however requires performing multiple stochastic forward passes at training time, for each minibatch, which could cause slower convergence to an optimal policy as observed with BBQN with α-divergences.

Other future work which was pointed out to previously includes using a mixture density network or a heavy tailed distribution instead of a Gaussian when minimizing the log-likelihood. The use of ensembles and adversarial training can improve uncertainty estimates as well.

In addition to the previous potential improvements, existing methods that has proved to improve DQN's performance could be applied to BBQN. For instance, prioritized experience replay (PER) [40] can be employed instead of experience replay. PER prioritizes transitions that can make experience replay more efficient and effective than if all transitions are replayed uniformly. PER allows to decorrelate updates that can occur from processing transition in the exact order they are experienced, and liberates agents from considering transitions with the same frequency that they are experienced. In [40], DQN with PER outperformed DQN with uniform replay on 41 out of 49 Atari games.

In this work, the parameters for initializing the weights in the neural networks and for determining the variance of the prior and the variance of the sampling distribution in BBQN were set using a grid search. It is unlikely that these are optimal, and a better hyperparameter tuning strategy would offer an improvement. An example of such strategy is the framework of Bayesian optimization, in which a learning algorithm's generalization performance is modeled as a sample from a GP [41]. The authors in [41] show that Bayesian optimization finds better hyperparameters significantly faster than other approaches for automatic tuning of hypeparameters for machine learning algorithms and surpasses a human expert at selecting hyperparameters for convolutional neural networks.

Future work should also evaluate BBQN on the master space and with real human users to further demonstrate its capabilities in terms of sample efficiency and computational time compared to GPSARSA and other deep RL methods.

Conclusion

This thesis has described how the Bayes By Backprop method can be applied successfully to obtain uncertainty estimates in DQN (BBQN), when applied to POMDP-based dialogue management. The results obtained confirm that BBQN learns dialogue policies with more efficient exploration than ε-greedy based methods, and reach performance comparable to the state of the art in policy optimization, namely GPSARSA, especially when evaluated on more complex domains. BBQN is also almost as sample efficient as GPSASA, but without the cubic computational complexity of GPs (or quadratic complexity when sparcification is used). When trained with a noise level of 15%, then evaluated across a range of noise levels from 0% to 45%, BBQN achieved higher performance at higher confusion rates than other deep RL methods, and its performance decreased the least with increasing confusion rate at testing. This shows that BBQN generalizes better than other deep RL methods and is as robust as GPSARSA.

We also attempted to improve performance of BBQN by using α-divergences or by using different methods for obtaining uncertainty estimates, such as dropout, and minimizing the NLL. Using α-divergences or minimizing the NLL did not make any improvements to the average reward and success rate of BBQN. On the other hand, variational dropout achieved a more stable performance at later stages of training compared to BBQN, and is a promising avenue to explore. Future work can focus on using better methods to tune the dropout probability. One method is to continuously relax the dropout's discrete masks and optimize the dropout probability using gradient methods.

Future research in this area will need to address a number of issues. First, improvements to the uncertainty of estimates for BBQN are needed to improve its sample efficiency. A better stability at later stages of training needs also to be addressed with BBQN. Methods for more automatic and better hyperparameter tuning need to be considered as well. Implementing prioritized experience replay have the potential to improve performance of BBQN, and evaluation on the master space is worth considering to further demonstrate the benefits of BBQN.

Bibliography

[1] David Barber and Christopher M Bishop. "Ensemble learning in Bayesian neural networks". In: *NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES* 168 (1998), pp. 215–238.

[2] Charles Blundell et al. "Weight uncertainty in neural networks". In: *arXiv preprint arXiv:1505.05424* (2015).

[3] Yun-Nung Chen et al. "End-to-End Memory Networks with Knowledge Carryover for Multi-Turn Spoken Language Understanding." In: *INTERSPEECH*. 2016, pp. 3245–3249.

[4] Lucie Daubigney, Matthieu Geist, and Olivier Pietquin. "Off-policy learning in large-scale POMDP-based dialogue systems". In: *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE. 2012, pp. 4989–4992.

[5] Lucie Daubigney et al. "Uncertainty management for on-line optimisation of a POMDP-based large-scale spoken dialogue system". In: *Interspeech 2011*. 2011, pp. 1301–1304.

[6] Stefan Depeweg et al. "Learning and policy search in stochastic dynamical systems with bayesian neural networks". In: *arXiv preprint arXiv:1605.07127* (2016).

[7] Mehdi Fatemi et al. "Policy networks with two-stage training for dialogue systems". In: *arXiv preprint arXiv:1606.03152* (2016).

[8] Meire Fortunato et al. "Noisy Networks for Exploration". In: *arXiv preprint arXiv:1706.10295* (2017).

[9] Yarin Gal. "Uncertainty in deep learning". PhD thesis. PhD thesis, University of Cambridge, 2016.

[10] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. 2016, pp. 1050–1059.

[11] Yarin Gal, Jiri Hron, and Alex Kendall. "Concrete Dropout". In: *arXiv preprint arXiv:1705.07832* (2017).

[12] Milica Gasic and Steve Young. "Gaussian processes for pomdp-based dialogue manager optimization". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.1 (2014), pp. 28–40.

[13] Milica Gašić et al. "Dialogue manager domain adaptation using Gaussian process reinforcement learning". In: *Computer Speech & Language* 45 (2017), pp. 552–569.

[14] Milica Gašić et al. "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*. Association for Computational Linguistics. 2010, pp. 201–204.

[15] Mohammad Ghavamzadeh et al. “Bayesian reinforcement learning: A survey”. In: *Foundations and Trends® in Machine Learning* 8.5-6 (2015), pp. 359–483.

[16] Alex Graves. “Practical variational inference for neural networks”. In: *Advances in Neural Information Processing Systems*. 2011, pp. 2348–2356.

[17] Matthew S Henderson. “Discriminative methods for statistical spoken dialogue systems”. PhD thesis. University of Cambridge, 2015.

[18] José Miguel Hernández-Lobato et al. “Black-box α-divergence minimization”. In: (2016).

[19] 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*. ACM. 1993, pp. 5–13.

[20] Rein Houthooft et al. “Variational information maximizing exploration”. In: *Advances in Neural Information Processing Systems (NIPS)* (2016).

[21] Rudolf Kadlec et al. “IBM’s belief tracker: Results on dialog state tracking challenge datasets”. In: *EACL 2014* (2014), p. 10.

[22] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1 (1998), pp. 99–134.

[23] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[24] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[25] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. “Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles”. In: *arXiv preprint arXiv:1612.01474* (2016).

[26] Yingzhen Li and Yarin Gal. “Dropout Inference in Bayesian Neural Networks with Alpha-divergences”. In: *arXiv preprint arXiv:1703.02914* (2017).

[27] Zachary C Lipton et al. “Efficient exploration for dialogue policy learning with BBQ networks & replay buffer spiking”. In: *arXiv preprint arXiv:1608.05081* (2016).

[28] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* 4.3 (1992), pp. 448–472.

[29] François Mairesse and Steve Young. “Stochastic language generation in dialogue using factored language models”. In: *Computational Linguistics* (2014).

[30] Thomas P Minka. “Expectation propagation for approximate Bayesian inference”. In: *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2001, pp. 362–369.

[31] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[32] Radford M Neal. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012.

[33] David A Nix and Andreas S Weigend. “Estimating the mean and variance of the target probability distribution”. In: *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference On*. Vol. 1. IEEE. 1994, pp. 55–60.

[34] Aaron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

[35] Ian Osband et al. “Deep exploration via bootstrapped DQN”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4026–4034.

[36] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. “Natural actor-critic”. In: *European Conference on Machine Learning*. Springer. 2005, pp. 280–291.

[37] Joaquin Quiaonero-Candela, Carl Edward Rasmussen, AnÃbal R Figueiras-Vidal, et al. “Sparse spectrum Gaussian process regression”. In: *Journal of Machine Learning Research* 11.Jun (2010), pp. 1865–1881.

[38] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*. Vol. 1. MIT press Cambridge, 2006.

[39] Jost Schatzmann et al. “A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies”. In: *The knowledge engineering review* 21.2 (2006), pp. 97–126.

[40] Tom Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015).

[41] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[42] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[43] Pei-Hao Su et al. “Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management”. In: *arXiv preprint arXiv:1707.00130* (2017).

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

[45] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[46] William R Thompson. “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples”. In: *Biometrika* 25.3/4 (1933), pp. 285–294.

[47] Stefan Ultes et al. “PyDial: A Multi-domain Statistical Dialogue System Toolkit”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics–System Demonstrations*. Vancouver: ACL, Aug. 2017.

[48] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning.” In: *AAAI*. 2016, pp. 2094–2100.

[49] Jason D Williams and Steve Young. “Partially observable Markov decision processes for spoken dialog systems”. In: *Computer Speech & Language* 21.2 (2007), pp. 393–422.

[50] Steve J Young. “Talking to machines (statistically speaking).” In: *INTERSPEECH*. 2002.

[51] Steve Young et al. “Pomdp-based statistical spoken dialog systems: A review”. In: *Proceedings of the IEEE* 101.5 (2013), pp. 1160–1179.

[52] Steve Young et al. “The hidden information state approach to dialog management”. In: *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*. Vol. 4. IEEE. 2007, pp. IV–149.

Appendix A

Weighing the $\mathcal{KL}$ divergence

In this appendix, we show the values for the $\mathcal{KL}$ divergence term and the loglikelihood term in the loss function for BBQN, as a function of the training dialogues for different approaches for weighing the $\mathcal{KL}$ divergence.

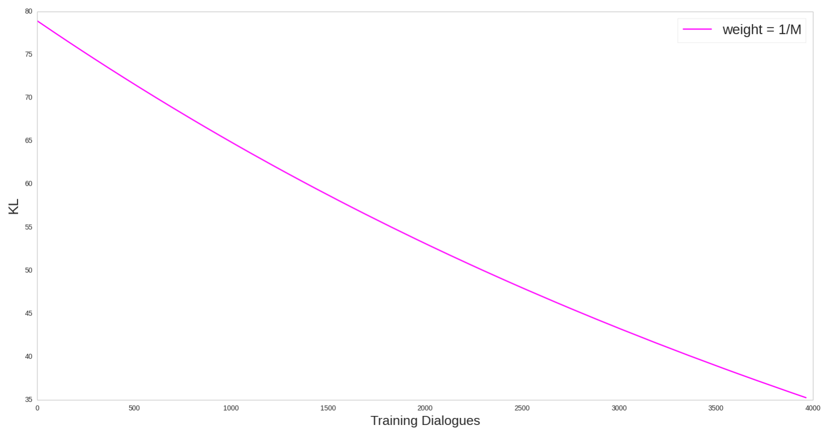


Figure A.1: $\mathcal{KL}$-divergence as a function of the training dialogues, for $\xi_i = \frac{1}{M}$.

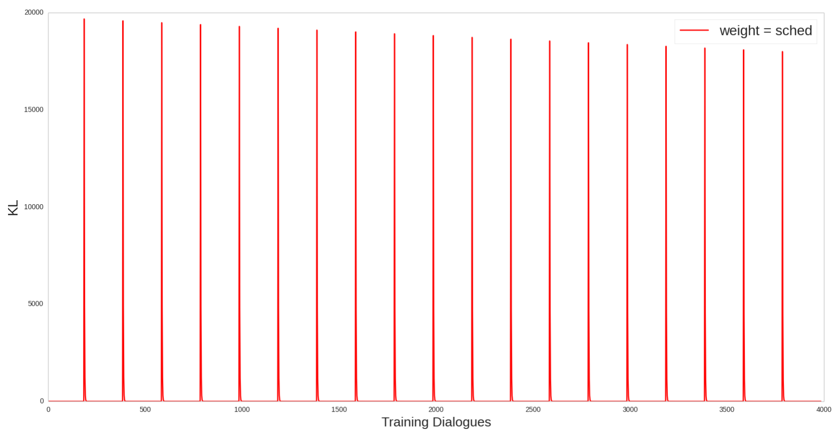


Figure A.2: $\mathcal{KL}$-divergence as a function of the training dialogues, for $\xi_i = \frac{2^{M-i}}{2^M - 1}$.

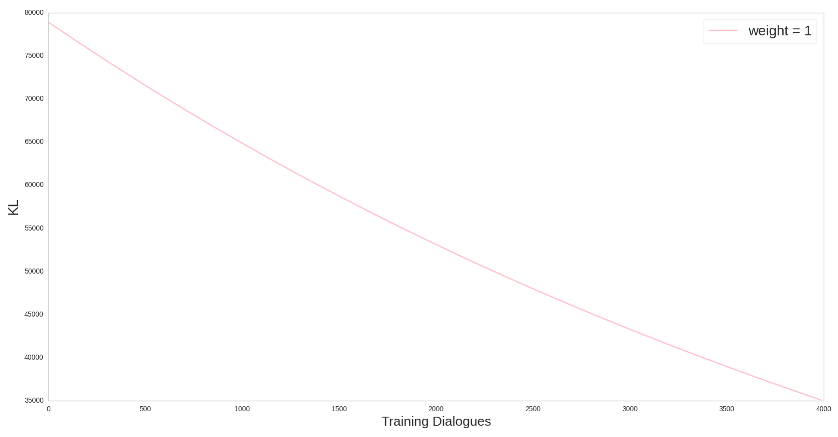


Figure A.3: $\mathcal{KL}$-divergence as a function of the training dialogues, for $\xi_i = 1$.

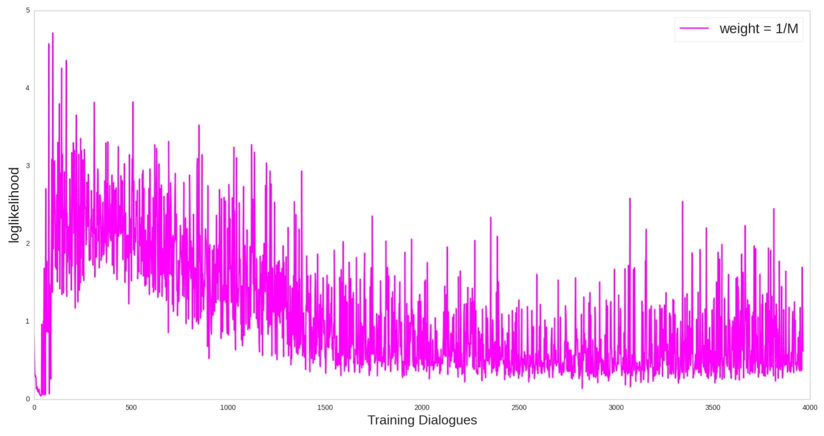


Figure A.4: Loglikelihood as a function of the training dialogues, for $\xi_i = \frac{1}{M}$.

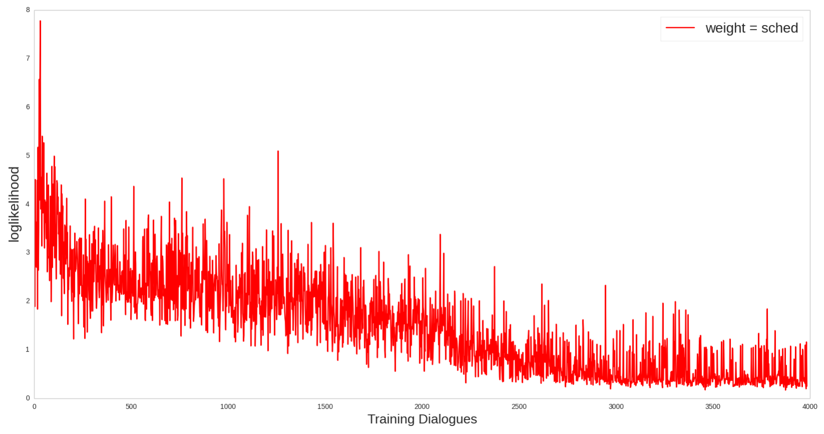


Figure A.5: Loglikelihood as a function of the training dialogues, for $\xi_i = \frac{2^{M-i}}{2^M-1}$.

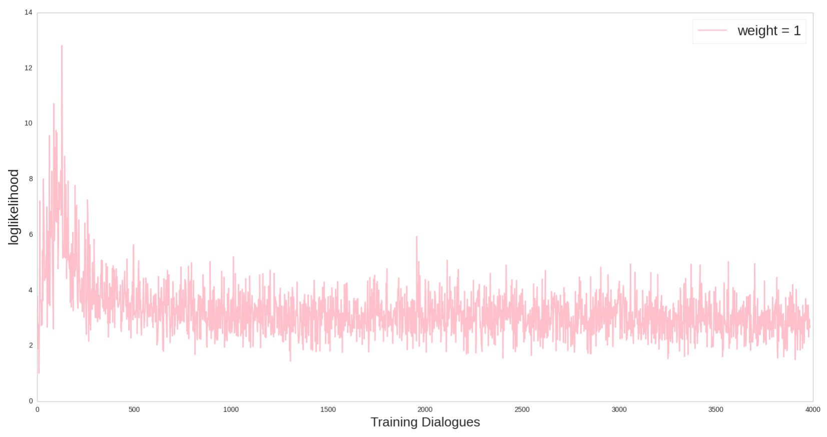


Figure A.6: Loglikelihood as a function of the training dialogues, for $\xi_i = 1$.