# Policy Networks with Two-Stage Training for Dialogue Systems

**Mehdi Fatemi** **Layla El Asri** **Hannes Schulz** **Jing He** **Kaheer Suleman**

Maluuba Research
Le 2000 Peel, Montréal, QC H3A 2W5
first.last@maluuba.com

## Abstract

In this paper, we propose to use deep policy networks which are trained with an advantage actor-critic method for statistically optimised dialogue systems. First, we show that, on summary state and action spaces, deep Reinforcement Learning (RL) outperforms Gaussian Processes methods. Summary state and action spaces lead to good performance but require pre-engineering effort, RL knowledge, and domain expertise. In order to remove the need to define such summary spaces, we show that deep RL can also be trained efficiently on the original state and action spaces. Dialogue systems based on partially observable Markov decision processes are known to require many dialogues to train, which makes them unappealing for practical deployment. We show that a deep RL method based on an actor-critic architecture can exploit a small amount of data very efficiently. Indeed, with only a few hundred dialogues collected with a handcrafted policy, the actor-critic deep learner is considerably bootstrapped from a combination of supervised and batch RL. In addition, convergence to an optimal policy is significantly sped up compared to other deep RL methods initialized on the data with batch RL. All experiments are performed on a restaurant domain derived from the Dialogue State Tracking Challenge 2 (DSTC2) dataset.

## 1 Introduction

The statistical optimization of dialogue management in dialogue systems through Reinforcement Learning (RL) has been an active thread of research for more than two decades (Levin et al., 1997; Lemon and Pietquin, 2007; Laroche et al., 2010; Gašić et al., 2012; Daubigney et al., 2012). Dialogue management has been successfully modelled as a Partially Observable Markov Decision Process (POMDP) (Williams and Young, 2007; Gašić et al., 2012), which leads to systems that can learn from data and which are robust to noise. In this context, a dialogue between a user and a dialogue system is framed as a sequential process where, at each turn, the system has to act based on what it has understood so far of the user's utterances.

Unfortunately, POMDP-based dialogue managers have been unfit for online deployment because they typically require several thousands of dialogues for training (Gašić et al., 2010, 2012). Nevertheless, recent work has shown that it is possible to train a POMDP-based dialogue system on just a few hundred dialogues corresponding to online interactions with users (Gašić et al., 2013). However, in order to do so, pre-engineering efforts, prior RL knowledge, and domain expertise must be applied. Indeed, summary state and action spaces must be used and the set of actions must be restricted depending on the current state so that notoriously bad actions are prohibited.

In order to alleviate the need for a summary state space, deep RL (Mnih et al., 2013) has recently been applied to dialogue management (Cuayáhuitl et al., 2015) in the context of negotiations. It was shown that deep RL performed significantly better than other heuristic or supervised approaches. The authors performed learning over a large action space of 70 actions and they also had to use restricted action sets in order to learn efficiently over this space. Besides, deep RL was not compared to other RL methods, which we do in this paper. In (Cuayáhuitl, 2016), a simplistic implementation of deep Q Networks is presented,



*Proceedings of the SIGDIAL 2016 Conference*, pages 101–110,
Los Angeles, USA, 13-15 September 2016. 

again with no comparison to other RL methods.

In this paper, we propose to efficiently alleviate the need for summary spaces and restricted actions using deep RL. We analyse four deep RL models: Deep Q Networks (DQN) (Mnih et al., 2013), Double DQN (DDQN) (van Hasselt et al., 2015), Deep Advantage Actor-Critic (DA2C) (Sutton et al., 2000) and a version of DA2C initialized with supervised learning (TDA2C)[1] (similar idea to Silver et al. (2016)). All models are trained on a restaurant-seeking domain. We use the Dialogue State Tracking Challenge 2 (DSTC2) dataset to train an agenda-based user simulator (Schatzmann and Young, 2009) for online learning and to perform batch RL and supervised learning.

We first show that, on summary state and action spaces, deep RL converges faster than Gaussian Processes SARSA (GPSARSA) (Gašić et al., 2010). Then we show that deep RL enables us to work on the original state and action spaces. Although GPSARSA has also been tried on original state space (Gašić et al., 2012), it is extremely slow in terms of wall-clock time due to its growing kernel evaluations. Indeed, contrary to methods such as GPSARSA, deep RL performs efficient generalization over the state space and memory requirements do not increase with the number of experiments. On the simple domain specified by DSTC2, we do not need to restrict the actions in order to learn efficiently. In order to remove the need for restricted actions in more complex domains, we advocate for the use of TDA2C and supervised learning as a pre-training step. We show that supervised learning on a small set of dialogues (only 706 dialogues) significantly bootstraps TDA2C and enables us to start learning with a policy that already selects only valid actions, which makes for a safe user experience in deployment. Therefore, we conclude that TDA2C is very appealing for the practical deployment of POMDP-based dialogue systems.

In Section 2 we briefly review POMDP, RL and GPSARSA. The value-based deep RL models investigated in this paper (DQN and DDQN) are described in Section 3. Policy networks and DA2C are discussed in Section 4. We then introduce the two-stage training of DA2C in Section 5. Experimental results are presented in Section 6. Finally, Section 7 concludes the paper and makes suggestions for future research.

[1]Teacher DA2C

## 2 Preliminaries

The reinforcement learning problem consists of an environment (the user) and an agent (the system) (Sutton and Barto, 1998). The environment is described as a set of continuous or discrete states $\mathcal{S}$ and at each state $s \in \mathcal{S}$, the system can perform an action from an action space $\mathcal{A}(s)$. The actions can be continuous, but in our case they are assumed to be discrete and finite. At time $t$, as a consequence of an action $A_t = a \in \mathcal{A}(s)$, the state transitions from $S_t = s$ to $S_{t+1} = s' \in \mathcal{S}$. In addition, a reward signal $R_{t+1} = R(S_t, A_t, S_{t+1}) \in \mathbb{R}$ provides feedback on the quality of the transition[2]. The agent's task is to maximize at each state the expected discounted sum of rewards received after visiting this state. For this purpose, value functions are computed. The action-state value function $Q$ is defined as:

$$Q^\pi(S_t, A_t) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s, A_t = a], \quad (1)$$

where $\gamma$ is a discount factor in $[0, 1]$. In this equation, the *policy* $\pi$ specifies the system's behaviour, *i.e.*, it describes the agent's action selection process at each state. A policy can be a deterministic mapping $\pi(s) = a$, which specifies the action $a$ to be selected when state $s$ is met. On the other hand, a stochastic policy provides a probability distribution over the action space at each state:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]. \quad (2)$$

The agent's goal is to find a policy that maximizes the $Q$-function at each state.

It is important to note that here the system does not have direct access to the state $s$. Instead, it sees this state through a *perception* process which typically includes an *Automatic Speech Recognition* (ASR) step, a *Natural Language Understanding* (NLU) step, and a *State Tracking* (ST) step. This perception process injects noise in the state of the system and it has been shown that modelling dialogue management as a POMDP helps to overcome this noise (Williams and Young, 2007; Young et al., 2013).

Within the POMDP framework, the state at time $t$, $S_t$, is not directly observable. Instead, the system has access to a noisy observation $O_t$.[3] A

[2]In this paper, upper-case letters are used for random variables, lower-case letters for non-random values (known or unknown), and calligraphy letters for sets.

[3]Here, the representation of the user's goal and the user's utterances.

POMDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, Z, \gamma, b_0)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P$ is the function encoding the transition probability: $P_a(s, s') = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$, $R$ is the reward function, $\mathcal{O}$ is the observation space, $Z$ encodes the observation probabilities $Z_a(s, o) = \mathbb{P}(O_t = o \mid S_t = s, A_t = a)$, $\gamma$ is a discount factor, and $b_0$ is an initial belief state. The *belief state* is a distribution over states. Starting from $b_0$, the state tracker maintains and updates the belief state according to the observations perceived during the dialogue. The dialogue manager then operates on this belief state. Consequently, the value functions as well as the policy of the agent are computed on the belief states $B_t$:

$$Q^\pi(B_t, A_t) = \mathbb{E}_\pi \left[ \sum_{t' \geq t} \gamma^{t'-t} R_{t'+1} \mid B_t, A_t \right]$$
$$\pi(a|b) = \mathbb{P}[A_t = a | B_t = b]. \quad (3)$$

In this paper, we use GPSARSA as a baseline as it has been proved to be a successful algorithm for training POMDP-based dialogue managers (Engel et al., 2005; Gašić et al., 2010). Formally, the $Q$-function is modelled as a Gaussian process, entirely defined by a mean and a kernel: $Q(B, A) \sim \mathcal{GP}(m, (k(B, A), k(B, A)))$. The mean is usually initialized at 0 and it is then jointly updated with the covariance based on the system's observations (*i.e.*, the visited belief states and actions, and the rewards). In order to avoid intractability in the number of experiments, we use kernel span sparsification (Engel et al., 2005). This technique consists of approximating the kernel on a dictionary of linearly independent belief states. This dictionary is incrementally built during learning. Kernel span sparsification requires setting a threshold on the precision to which the kernel is computed. As discussed in Section 6, this threshold needs to be fine-tuned for a good tradeoff between precision and performance.

## 3 Value-Based Deep Reinforcement Learning

Broadly speaking, there are two main streams of methodologies in the RL literature: value approximation and policy gradients. As suggested by their names, the former tries to approximate the value function whereas the latter tries to directly approximate the policy. Approximations are necessary for large or continuous belief and action spaces. Indeed, if the belief space is large or continuous it would not be possible to store a value for each state in a table, so generalization over the state space is necessary. In this context, some of the benefits of deep RL techniques are the following:

- Generalisation over the belief space is efficient and the need for summary spaces is eliminated, normally with considerably less wall-clock training time comparing to GP-SARSA, for example.

- Memory requirements are limited and can be determined in advance unlike with methods such as GPSARSA.

- Deep architectures with several hidden layers can be efficiently used for complex tasks and environments.

### 3.1 Deep Q Networks

A Deep $Q$-Network (DQN) is a multi-layer neural network which maps a belief state $B_t$ to the values of the possible actions $A_t \in \mathcal{A}(B_t = b)$ at that state, $Q^\pi(B_t, A_t;\ w_t)$, where $w_t$ is the weight vector of the neural network. Neural networks for the approximation of value functions have long been investigated (Bertsekas and Tsitsiklis, 1996). However, these methods were previously quite unstable (Mnih et al., 2013). In DQN, Mnih et al. (2013, 2015) proposed two techniques to overcome this instability-namely *experience replay* and the use of a *target network*. In experience replay, all the transitions are put in a finite pool $\mathcal{D}$ (Lin, 1993). Once the pool has reached its predefined maximum size, adding a new transition results in deleting the oldest transition in the pool. During training, a mini-batch of transitions is *uniformly* sampled from the pool, *i.e.* $(B_t, A_t, R_{t+1}, B_{t+1}) \sim U(\mathcal{D})$. This method removes the instability arising from strong correlation between the subsequent transitions of an episode (a dialogue). Additionally, a target network with weight vector $w^-$ is used. This target network is similar to the $Q$-network except that its weights are only copied every $\tau$ steps from the $Q$-network, and remain fixed during all the other steps. The loss function for the $Q$-network at iter-

ation $t$ takes the following form:

$$L_t(w_t) = \mathbb{E}_{(B_t,A_t,R_{t+1},B_{t+1})\sim U(\mathcal{D})}\Big[ \Big(R_{t+1} + \gamma \max_{a'} Q^{\pi}(B_{t+1}, a'; w_t^-) - Q^{\pi}(B_t, A_t; w_t)\Big)^2 \Big]. \quad (4)$$

### 3.2 Double DQN: Overcoming Overestimation and Instability of DQN

The *max* operator in Equation 4 uses the same value network (*i.e.*, the target network) to select actions and evaluate them. This increases the probability of overestimating the value of the state-action pairs (van Hasselt, 2010; van Hasselt et al., 2015). To see this more clearly, the target part of the loss in Equation 4 can be rewritten as follows:

$$R_{t+1} + \gamma Q^{\pi}(B_{t+1}, \operatorname*{argmax}_a Q^{\pi}(B_{t+1}, a; w_t^-); w_t^-).$$

In this equation, the target network is used twice. Decoupling is possible by using the $Q$-network for action selection as follows (van Hasselt et al., 2015):

$$R_{t+1} + \gamma Q^{\pi}(B_{t+1}, \operatorname*{argmax}_a Q^{\pi}(B_{t+1}, a; w_t); w_t^-).$$

Then, similarly to DQN, the $Q$-network is trained using experience replay and the target network is updated every $\tau$ steps. This new version of DQN, called Double DQN (DDQN), uses the two value networks in a decoupled manner, and alleviates the overestimation issue of DQN. This generally results in a more stable learning process (van Hasselt et al., 2015).

In the following section, we present deep RL models which perform policy search and output a stochastic policy rather than value approximation with a deterministic policy.

## 4 Policy Networks and Deep Advantage Actor-Critic (DA2C)

A policy network is a parametrized probabilistic mapping between belief and action spaces:

$$\pi_\theta(a|b) = \pi(a|b;\theta) = \mathbb{P}(A_t = a|B_t = b, \theta_t = \theta),$$

where $\theta$ is the parameter vector (the weight vector of a neural network).[4] In order to train policy networks, policy gradient algorithms have been developed (Williams, 1992; Sutton et al., 2000). Policy gradient algorithms are model-free methods which directly approximate the policy by parametrizing it. The parameters are learnt using a gradient-based optimization method.

We first need to define an objective function $J$ that will lead the search for the parameters $\theta$. This objective function defines policy quality. One way of defining it is to take the average over the rewards received by the agent. Another way is to compute the discounted sum of rewards for each trajectory, given that there is a designated start state. The policy gradient is then computed according to the *Policy Gradient Theorem* (Sutton et al., 2000).

**Theorem 1 (Policy Gradient)** *For any differentiable policy $\pi_\theta(b, a)$ and for the average reward or the start-state objective function, the policy gradient can be computed as*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|b) Q^{\pi_\theta}(b, a)]. \quad (5)$$

Policy gradient methods have been used successfully in different domains. Two recent examples are AlphaGo by DeepMind (Silver et al., 2016) and MazeBase by Facebook AI (Sukhbaatar et al., 2016).

One way to exploit Theorem 1 is to parametrize $Q^{\pi_\theta}(b, a)$ separately (with a parameter vector $w$) and learn the parameter vector during training in a similar way as in DQN. The trained $Q$-network can then be used for policy evaluation in Equation 5. Such algorithms are known in general as *actor-critic* algorithms, where the $Q$ approximator is the critic and $\pi_\theta$ is the actor (Sutton, 1984; Barto et al., 1990; Bhatnagar et al., 2009). This can be achieved with *two* separate deep neural networks: a *$Q$-Network* and a *policy network*.

However, a direct use of Equation 5 with $Q$ as critic is known to cause high variance (Williams, 1992). An important property of Equation 5 can be used in order to overcome this issue: subtracting any differentiable function $Ba$ expressed over the belief space from $Q^{\pi_\theta}$ will not change the gradient. A good selection of $Ba$, which is called the *baseline*, can reduce the variance dramatically (Sutton and Barto, 1998). As a result, Equation 5 may be rewritten as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|b) Ad(b, a)], \quad (6)$$

[4] For parametrization, we use $w$ for value networks and $\theta$ for policy networks.

where $Ad(b,a) = Q^{\pi_\theta}(b,a) - Ba(b)$ is called the *advantage function*. A good baseline is the value function $V^{\pi_\theta}$, for which the advantage function becomes $Ad(b,a) = Q^{\pi_\theta}(b,a) - V^{\pi_\theta}(b)$. However, in this setting, we need to train two separate networks to parametrize $Q^{\pi_\theta}$ and $V^{\pi_\theta}$. A better approach is to use the TD error $\delta = R_{t+1} + \gamma V^{\pi_\theta}(B_{t+1}) - V^{\pi_\theta}(B_t)$ as advantage function. It can be proved that the expected value of the TD error is $Q^{\pi_\theta}(b,a) - V^{\pi_\theta}(b)$. If the TD error is used, only one network is needed, to parametrize $V^{\pi_\theta}(B_t) = V^{\pi_\theta}(B_t; w_t)$. We call this network the *value network*. We can use a DQN-like method to train the value network using both experience replay and a target network. For a transition $B_t = b$, $A_t = a$, $R_{t+1} = r$ and $B_{t+1} = b'$, the advantage function is calculated as in:

$$\delta_t = r + \gamma V^{\pi_\theta}(b'; w_t) - V^{\pi_\theta}(b; w_t). \quad (7)$$

Because the gradient in Equation 6 is weighted by the advantage function, it may become quite large. In fact, the advantage function may act as a large learning rate. This can cause the learning process to become unstable. To avoid this issue, we add $L_2$ regularization to the policy objective function. We call this method Deep Advantage Actor-Critic (DA2C).

In the next section, we show how this architecture can be used to efficiently exploit a small set of handcrafted data.

## 5 Two-stage Training of the Policy Network

By definition, the policy network provides a probability distribution over the action space. As a result and in contrast to value-based methods such as DQN, a policy network can also be trained with direct *supervised learning* (Silver et al., 2016). Supervised training of RL agents has been well-studied in the context of Imitation Learning (IL). In IL, an agent learns to reproduce the behaviour of an expert. Supervised learning of the policy was one of the first techniques used to solve this problem (Pomerleau, 1989; Amit and Mataric, 2002). This direct type of imitation learning requires that the learning agent and the expert share the same characteristics. If this condition is not met, IL can be done at the level of the value functions rather than the policy directly (Piot et al., 2015). In this paper, the data that we use (DSTC2) was collected with a dialogue system similar to the one we train so in our case, the demonstrator and the learner share the same characteristics.

Similarly to Silver et al. (2016), here, we initialize both the policy network and the value network on the data. The policy network is trained by minimising the categorical cross-entropy between the predicted action distribution and the demonstrated actions. The value network is trained directly through RL rather than IL to give more flexibility in the kind of data we can use. Indeed, our goal is to collect a small number of dialogues and learn from them. IL usually assumes that the data corresponds to expert policies. However, dialogues collected with a handcrafted policy or in a Wizard-of-Oz (WoZ) setting often contain both optimal and sub-optimal dialogues and RL can be used to learn from all of these dialogues. Supervised training can also be done on these dialogues as we show in Section 6.

Supervised actor-critic architectures following this idea have been proposed in the past (Benbrahim and Franklin, 1997; Si et al., 2004); the actor works together with a human supervisor to gain competence on its task even if the critic's estimations are poor. For instance, a human can help a robot move by providing the robot with valid actions. We advocate for the same kind of methods for dialogue systems. It is easy to collect a small number of high-quality dialogues and then use supervised learning on this data to teach the system valid actions. This also eliminates the need to define restricted action sets.

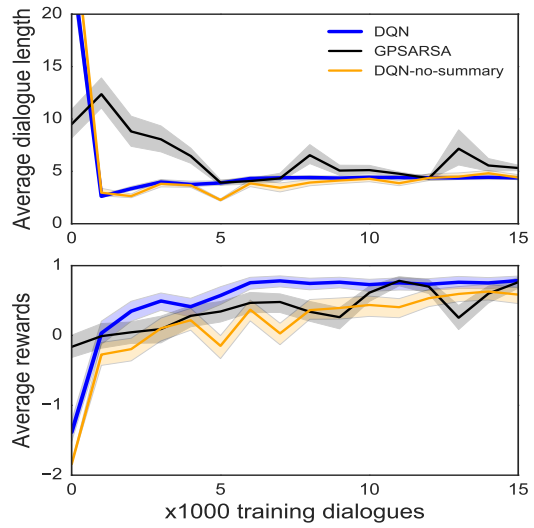
In all the methods above, *Adadelta* will be used as the gradient-decent optimiser, which in our experiments works noticeably better than other methods such as *Adagrad*, *Adam*, and *RMSProp*.

## 6 Experiments

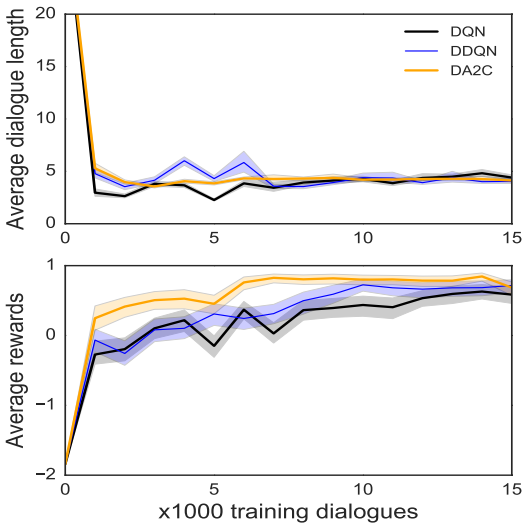
### 6.1 Comparison of DQN and GPSARSA

#### 6.1.1 Experimental Protocol

In this section, as a first argument in favour of deep RL, we perform a comparison between GPSARSA and DQN on simulated dialogues. We trained an agenda-based user simulator which at each dialogue turn, provides one or several dialogue act(s) in response to the latest machine act (Schatzmann et al., 2007; Schatzmann and Young, 2009). The dataset used for training this user-simulator is the Dialogue State Tracking Challenge 2 (DSTC2) (Henderson et al., 2014) dataset. State tracking is also trained on this dataset. DSTC2 includes




(a) Comparison of GPSARSA on summary spaces and DQN on summary (DQN) and original spaces (DQN-no-summary).




(b) Comparison of DA2C, DQN and DDQN on original spaces.

Figure 1: Comparison of different algorithms on simulated dialogues, without any pre-training.

dialogues with users who are searching for restaurants in Cambridge, UK.

In each dialogue, the user has a goal containing constraint slots and request slots. The constraint and request slots available in DSTC2 are listed in Appendix A. The constraints are the slots that the user has to provide to the system (for instance the user is looking for a specific type of food in a given area) and the requests are the slots that the user must receive from the system (for instance the user wants to know the address and phone number of the restaurant found by the system).

Similarly, the belief state is composed of two parts: constraints and requests. The constraint part includes the probabilities of the top two values for each constraint slot as returned by the state tracker (the value might be empty with a probability zero if the slot has not been mentioned). The request part, on the other hand, includes the probability of each request slot. For instance the constraint part might be [food: (Italian, 0.85) (Indian, 0.1) (Not_mentioned, 0.05)] and the request part might be [area: 0.95] meaning that the user is probably looking for an Italian restaurant and that he wants to know the area of the restaurant found by the system. To compare DQN to GPSARSA, we work on a summary state space (Gašić et al., 2012, 2013). Each constraint is mapped to a one-hot vector, with 1 corresponding to the tuple in the grid vector $g_c = [(1,0),(.8,.2),(.6,.2),(.6,.4),(.4,.4)]$ that minimizes the Euclidean distance to the top two probabilities. Similarly, each request slot is mapped to a one-hot vector according to the grid $g_r = [1,.8,.6,.4,0.]$. The final belief vector, known as the summary state, is defined as the concatenation of the constraint and request one-hot vectors. Each summary state is a binary vector of length 60 (12 one-hot vectors of length 5) and the total number of states is $5^{12}$.

We also work on a summary action space and we use the act types listed in Table 1 in Appendix A. We add the necessary slot information as a post processing step. For example, the *request* act means that the system wants to request a slot from the user, *e.g.* request(food). In this case, the selection of the slot is based on min-max probability, *i.e.*, the most ambiguous slot (which is the slot we want to request) is assumed to be the one for which the value with maximum probability has the minimum probability compared to the most certain values of the other slots. Note that this heuristic approach to compute the summary state and action spaces is a requirement to make GPSARSA tractable; it is a serious limitation in general and should be avoided.

As reward, we use a normalized scheme with a reward of +1 if the dialogue finishes successfully

before 30 turns,[5] a reward of -1 if the dialogue is not successful after 30 turns, and a reward of -0.03 for each turn. A reward of -1 is also distributed to the system if the user hangs up. In our settings, the user simulator hangs up every time the system proposes a restaurant which does not match at least one of his constraints.

For the deep $Q$-network, a Multi-Layer Perceptron (MLP) is used with two fully connected hidden layers, each having a *tanh* activation. The output layer has no activation and it provides the value for each of the summary machine acts. The summary machine acts are mapped to original acts using the heuristics explained previously. Both algorithms are trained with 15000 dialogues. GPSARSA is trained with $\epsilon$-*softmax* exploration, which, with probability $1 - \epsilon$, selects an action based on the logistic distribution $\mathbb{P}[a|b] = \frac{e^{Q(b,a)}}{\sum_{a'} e^{Q(b,a')}}$ and, with probability $\epsilon$, selects an action in a uniformly random way. From our experiments, this exploration scheme works best in terms of both convergence rate and variance. For DQN, we use a simple $\epsilon$-*greedy* exploration which, with probability $1 - \epsilon$ (same $\epsilon$ as above), uniformly selects an action and, with probability $\epsilon$, selects an action maximizing the $Q$-function. For both algorithms, $\epsilon$ is annealed to less than 0.1 over the course of training.

In a second experiment, we remove both summary state and action spaces for DQN, *i.e.*, we do not perform the Euclidean-distance mapping as before but instead work directly on the probabilities themselves. Additionally, the state is augmented with the probability (returned by the state tracker) of each user act (see Table 2 in Appendix A), the dialogue turn, and the number of results returned by the database (0 if there was no query). Consequently, the state consists of 31 continuous values and two discrete values. The original action space is composed of 11 actions: `offer`[6], `select-area`, `select-food`, `select-pricerange`, `request-area`, `request-food`, `request-pricerange`, `expl-conf-area`, `expl-conf-food`, `expl-conf-pricerange`, `repeat`. There is no post-processing via min-max selection anymore since the slot is part of the action, *e.g.*, `select-area`.

The policies are evaluated after each 1000 training dialogues on 500 test dialogues without exploration.

#### 6.1.2 Results

Figure 1 illustrates the performance of DQN compared to GPSARSA. In our experiments with GPSARSA we found that it was difficult to find a good tradeoff between precision and efficiency. Indeed, for low precision, the algorithm learned rapidly but did not reach optimal behaviour, whereas higher precision made learning extremely slow but resulted in better end-performance. On summary spaces, DQN outperforms GPSARSA in terms of convergence. Indeed, GPSARSA requires twice as many dialogues to converge. It is also worth mentioning here that the wall-clock training time of GPSARSA is considerably longer than the one of DQN due to kernel evaluation. The second experiment validates the fact that Deep RL can be efficiently trained directly on the belief state returned by the state tracker. Indeed, DQN on the original spaces performs as well as GPSARSA on the summary spaces.

In the next section, we train and compare the deep RL networks previously described on the original state and action spaces.

### 6.2 Comparison of the Deep RL Methods

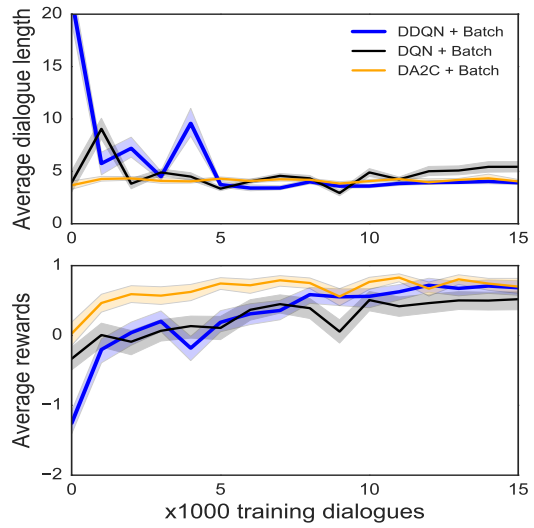
#### 6.2.1 Experimental Protocol

Similarly to the previous example, we work on a restaurant domain and use the DSTC2 specifications. We use $\epsilon-$greedy exploration for all four algorithms with $\epsilon$ starting at 0.5 and being linearly annealed at a rate of $\lambda = 0.99995$. To speed up the learning process, the actions `select-pricerange`, `select-area`, and `select-food` are excluded from exploration. Note that this set does not depend on the state and is meant for exploration only. All the actions can be performed by the system at any moment.

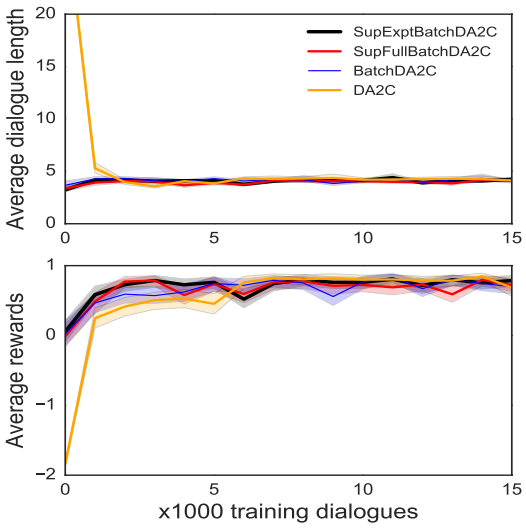
We derived two datasets from DSTC2. The first dataset contains the 2118 dialogues of DSTC2. We had these dialogues rated by a human expert, based on the quality of dialogue management and on a scale of 0 to 3. The second dataset only contains the dialogues with a rating of 3 (706 dialogues). The underlying assumption is that these dialogues correspond to optimal policies.

[5] A dialogue is successful if the user retrieves all the request slots for a restaurant matching all the constraints of his goal.

[6] This act consists of proposing a restaurant to the user. In order to be consistent with the DSTC2 dataset, an `offer` always contains the values for all the constraints understood by the system, *e.g.* offer(name = Super Ramen, food = Japanese, price range = cheap).




(a) Comparison of DA2C, DQN and DDQN after batch initialization.




(b) Comparison of DA2C and DA2C after batch initialization (batchDA2C), and TDA2C after supervised training on expert (SupExptBatchDA2C) and non-expert data (SupFullBatchDA2C).

Figure 2: Comparison of different algorithms on simulated dialogues, with pre-training.

We compare the convergence rates of the deep RL models in different settings. First, we compare DQN, DDQN and DA2C without any pre-training (Figure 1b). Then, we compare DQN, DDQN and TDA2C with an RL initialization on the DSTC2 dataset (Figure 2a). Finally, we focus on the advantage actor-critic models and compare DA2C, TDA2C, TDA2C with batch initialization on DSTC2, and TDA2C with batch initialization on the expert dialogues (Figure 2b).

#### 6.2.2 Results

As expected, DDQN converges faster than DQN on all experiments. Figure 1b shows that, without any pre-training, DA2C is the one which converges the fastest (6000 dialogues *vs.* 10000 dialogues for the other models). Figure 2a gives consistent results and shows that, with initial training on the 2118 dialogues of DSTC2, TDA2C converges significantly faster than the other models. Figure 2b focuses on DA2C and TDA2C. Compared to batch training, supervised training on DSTC2 speeds up convergence by 2000 dialogues (3000 dialogues *vs.* 5000 dialogues). Interestingly, there does not seem to be much difference between supervised training on the expert data and on DSTC2. The expert data only consists of 706 dialogues out of 2118 dialogues. Our observation is that, in the non-expert data, many of the dialogue acts chosen by the system were still appropriate, which explains that the system learns acceptable behavior from the entire dataset. This shows that supervised training, even when performed not only on optimal dialogues, makes learning much faster and relieves the need for restricted action sets. Valid actions are learnt from the dialogues and then RL exploits the good and bad dialogues to pursue training towards a high performing policy.

## 7 Concluding Remarks

In this paper, we used policy networks for dialogue systems and trained them in a two-stage fashion: supervised training and batch reinforcement learning followed by online reinforcement learning. An important feature of policy networks is that they directly provide a probability distribution over the action space, which enables supervised training. We compared the results with other deep reinforcement learning algorithms, namely Deep Q Networks and Double Deep Q Networks. The combination of supervised and reinforcement learning is the main benefit of our method, which paves the way for developing trainable end-to-end dialogue systems. Supervised training on a small dataset considerably bootstraps the learning process and can be used to significantly improve the

convergence rate of reinforcement learning in statistically optimised dialogue systems.

## References


R. Amit and M. Mataric. 2002. Learning movement sequences from demonstration. In *Proc. Int. Conf. on Development and Learning*. pages 203–208.

A. G. Barto, R. S. Sutton, and C. W. Anderson. 1990. In *Artificial Neural Networks*, chapter Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, pages 81–93.

H. Benbrahim and J. A. Franklin. 1997. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems* 22:283–302.

D. P. Bertsekas and J. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee. 2009. Natural Actor-Critic Algorithms. *Automatica* 45(11).

H. Cuayáhuitl. 2016. Simpleds: A simple deep reinforcement learning dialogue system. arXiv:1601.04574v1 [cs.AI].

H. Cuayáhuitl, S. Keizer, and O. Lemon. 2015. Strategic dialogue management via deep reinforcement learning. arXiv:1511.08099 [cs.AI].

L. Daubigney, M. Geist, S. Chandramohan, and O. Pietquin. 2012. A Comprehensive Reinforcement Learning Framework for Dialogue Management Optimisation. *IEEE Journal of Selected Topics in Signal Processing* 6(8):891–902.

Y. Engel, S. Mannor, and R. Meir. 2005. Reinforcement learning with gaussian processes. In *Proc. of ICML*.

M. Gašić, C. Breslin, M. Henderson, D. Kim, M. Szummer, B. Thomson, P. Tsiakoulis, and S.J. Young. 2013. On-line policy optimisation of bayesian spoken dialogue systems via human interaction. In *Proc. of ICASSP*. pages 8367–8371.

M. Gašić, M. Henderson, B. Thomson, P. Tsiakoulis, and S. Young. 2012. Policy optimisation of POMDP-based dialogue systems without state space compression. In *Proc. of SLT*.

M. Gašić, F. Jurčíček, S. Keizer, F. Mairesse, B. Thomson, K. Yu, and S. Young. 2010. Gaussian processes for fast policy optimisation of POMDP-based dialogue managers. In *Proc. of SIGDIAL*.

M. Henderson, B. Thomson, and J. Williams. 2014. The Second Dialog State Tracking Challenge. In *Proc. of SIGDIAL*.

R. Laroche, G. Putois, and P. Bretier. 2010. Optimising a handcrafted dialogue system design. In *Proc. of Interspeech*.

O. Lemon and O. Pietquin. 2007. Machine learning for spoken dialogue systems. In *Proc. of Interspeech*. pages 2685–2688.

E. Levin, R. Pieraccini, and W. Eckert. 1997. Learning dialogue strategies within the markov decision process framework. In *Proc. of ASRU*.

L-J Lin. 1993. *Reinforcement learning for robots using neural networks*. Ph.D. thesis, Carnegie Mellon University.

V Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing Atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*.

V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

B. Piot, M. Geist, and O. Pietquin. 2015. Imitation Learning Applied to Embodied Conversational Agents. In *Proc. of MLIS*.

D. A. Pomerleau. 1989. Alvinn: An autonomous land vehicle in a neural network. In *Proc. of NIPS*. pages 305–313.

J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young. 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Proc. of NAACL HLT*. pages 149–152.

J. Schatzmann and S. Young. 2009. The hidden agenda user simulation model. *Proc. of TASLP* 17(4):733–747.

J. Si, A. G. Barto, W. B. Powell, and D. Wunsch. 2004. *Supervised ActorCritic Reinforcement Learning*, pages 359–380.

D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser,

I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

S. Sukhbaatar, A. Szlam, G. Synnaeve, S. Chintala, and R. Fergus. 2016. Mazebase: A sandbox for learning from games. arxiv.org/pdf/1511.07401 [cs.LG].

R. S. Sutton. 1984. *Temporal credit assignment in reinforcement learning*. Ph.D. thesis, University of Massachusetts at Amherst, Amherst, MA, USA.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Proc. of NIPS*. volume 12, pages 1057–1063.

R.S. Sutton and A.G. Barto. 1998. *Reinforcement Learning*. MIT Press.

H. van Hasselt. 2010. Double q-learning. In *Proc. of NIPS*. pages 2613–2621.

H. van Hasselt, A. Guez, and D. Silver. 2015. Deep reinforcement learning with double Q-learning. arXiv:1509.06461v3 [cs.LG].

J.D. Williams and S. Young. 2007. Partially observable markov decision processes for spoken dialog systems. *Proc. of CSL* 21:231–422.

R.J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8:229–256.

S. Young, M. Gasic, B. Thomson, and J. Williams. 2013. POMDP-based statistical spoken dialog systems: A review. *Proc. IEEE* 101(5):1160–1179.

## A Specifications of restaurant search in DTSC2

**Constraint slots** area, type of food, price range.

**Request slots** area, type of food, address, name, price range, postcode, signature dish, phone number

Table 1: Summary actions.

| Action | Description |
|---|---|
| Cannot help | No restaurant in the database matches the user's constraints. |
| Confirm Domain | Confirm that the user is looking for a restaurant. |
| Explicit Confirm | Ask the user to confirm a piece of information. |
| Offer | Propose a restaurant to the user. |
| Repeat | Ask the user to repeat. |
| Request | Request a slot from the user. |
| Select | Ask the user to select a value between two propositions (*e.g.* select between Italian and Indian). |

Table 2: User actions.

| Action | Description |
|---|---|
| Deny | Deny a piece of information. |
| Null | Say nothing. |
| Request More | Request more options. |
| Confirm | Ask the system to confirm a piece of information. |
| Acknowledge | Acknowledge. |
| Affirm | Say yes. |
| Request | Request a slot value. |
| Inform | Inform the system of a slot value. |
| Thank you | Thank the system. |
| Repeat | Ask the system to repeat. |
| Request Alternatives | Request alternative restaurant options. |
| Negate | Say no. |
| Bye | Say goodbye to the system. |
| Hello | Say hello to the system. |
| Restart | Ask the system to restart the dialogue. |