

Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
Pfaffenwaldring 5B
D-70569 Stuttgart

Deep Reinforcement Learning in Dialog Systems

Dirk Väth
Master Thesis

Studiengang: Informatik
Prüfer & Betreuer: Prof. Dr. Thang Vu
Prüferin: Dr. Antje Schweitzer

Beginn der Arbeit: 01.06.2018
Ende der Arbeit: 30.11.2018

Abstract

This thesis explores advanced deep reinforcement learning methods for learning dialog policies. While many recent contributions in the area of reinforcement learning focus on learning how to play Atari games, this thesis applies them in a real-world scenario. When talking to a dialog system, the dialog policy is the component which chooses the response based on the history of the interaction between user and system. Nowadays, dialog policies may be learned automatically by training a reinforcement learning agent with a user simulator. In this thesis, a baseline method for dialog policy learning is implemented and extended by various state-of-the-art deep reinforcement learning methods. An ablation study discusses the significance of each extension, highlighting beneficial and harmful additions. Each extended agent is shown to perform better than the baseline method with all agents outperforming policies from an existing benchmark. Two agents even prove to be on par with handcrafted dialog policies. Along with the quantitative evaluation, qualitative results are provided in the form of chats between a user and a trained agent.

Zusammenfassung

Die vorliegende Masterarbeit untersucht fortgeschrittene Deep Reinforcement Learning Techniken zum Erlernen von Dialogstrategien. Während viele jüngst veröffentlichte Beiträge im Bereich Reinforcement Learning auf das Erlernen von Atari-Spielen fokussiert sind, werden sie hier auf ein realistisches Szenario angewendet. Spricht man mit einem Dialogsystem, ist die Dialogstrategie verantwortlich für die Auswahl der Antwort, basierend auf dem bisherigen Konversationsverlauf. Heutzutage können Dialogstrategien automatisiert erlernt werden, indem ein Reinforcement Learning Agent mit einem simulierten Nutzer trainiert wird. In dieser Arbeit wird eine Basismethode implementiert und mit verschiedenen aktuellen Reinforcement Learning Methoden erweitert. Durch das anschließende Entfernen einzelner Methoden

wird deren individueller positiver oder negativer Einfluss diskutiert. Es wird gezeigt, dass jeder erweiterte Agent bessere Ergebnisse erzielt als die Basismethode und auch bereits existierende Benchmarks schlägt. Zwei Agenten erzielen sogar Ergebnisse, die manuell erzeugten Strategien ebenbürtig sind. Zusammen mit der quantitativen Evaluation werden auch qualitative Ergebnisse in Form von aufgezeichneten Dialogen zwischen einem echten Nutzer und trainierten Agenten vorgestellt.

Contents

List of Figures	IV
List of Tables	V
Acronyms	VI
1 Introduction	1
2 Background	3
2.1 Spoken Dialog Systems	3
2.1.1 Dialog Policies	4
2.2 Reinforcement Learning	5
2.2.1 Agent-Environment-Interaction	5
2.2.2 Markov Decision Process	7
2.2.3 Value functions	7
2.2.4 Temporal Difference Algorithms	10
2.2.5 Q-Learning	10
2.3 Dialog Policies as Markov Decision Problems	11
2.4 Neural Networks	12
3 Related Work	14
3.1 Dialog Policy Learning	14
3.2 Deep Reinforcement Learning	15
3.2.1 Deep Q-Networks	16
3.2.2 Prioritized Experience Replay	16

3.2.3	Double DQN	18
3.2.4	Multi-Step Targets	19
3.2.5	Noisy Networks for Exploration	19
3.2.6	Dueling network architecture	21
3.2.7	Distributional DQN	23
3.2.8	Rainbow	25
4	Resources	26
4.1	OpenAI Gym	26
4.2	PyDial	27
5	Methods	28
5.1	Combined agent	28
5.1.1	Combined agent without dueling network architecture .	31
5.1.2	Combined agent without distributional learning method	31
5.1.3	Combined agent without double Q-learning	32
5.1.4	Combined agent without prioritized experience replay .	32
5.1.5	Combined agent with multi-step targets	32
5.1.6	Combined agent with noisy linear layers	32
6	Results and Discussion	33
6.1	Hyperparameters	33
6.2	Experiment setup	34
6.3	Results	34
6.4	Discussion	36
7	Conclusion and Future Work	44

Appendices	49
Bibliography	57

List of Figures

1	Schematic of a modular dialog system	4
2	Interaction between agent and environment.	6
3	Example MDP	9
4	Example feed-forward network	13
5	Schematic of a dueling network	22
6	Average rewards for the tested agents	38
7	Clustered Q-values for the combined agent	40
8	Clustered Q-values for the non-distributional agent	41
9	Rewards for the multi-step agent with step size 2 and the combined agent for different tasks.	50
10	Train loss of the multi-step agent for task T1.1 with step size 2.	54
11	Train loss of the multi-step agent for task T1.1 with step size 5.	54
12	Results for the combined agent on the OpenAI Gym cart-pole environment	55
13	Results for the combined agent on the OpenAI Gym cart-pole environment for noisy linear exploration	55

List of Tables

1	Benchmark ontologies	27
2	Benchmark tasks	28
3	Rewards for the tested agents	35
4	Success rates for the tested agents	36
5	Averaged results for the tested agents	37
6	Recorded dialog between a human and the trained combined agent for task T4.3.	39
7	Recorded dialog between a human and the best PyDial agent (DQN) for task T4.3.	47
8	Recorded dialog between a human and PyDial’s handcrafted policy for task T4.3.	48
9	Rewards for the combined agent with noisy linear layers for exploration	49
10	Success rates for the combined agent with noisy linear layers for exploration	51
11	Rewards for the combined agent with multi-step targets	52
12	Success rates for the combined agent with multi-step targets	53
13	Hyperparameters used for OpenAI Gym environment	56

Acronyms

ASR Automatic Speech Recognition.

CDF Cumulative Distribution Function.

DQN Deep Q-Networks.

DST Dialog State Tracker.

MDP Markov Decision Process.

NLG Natural Language Generator.

QR-DQN Quantile Regression DQN.

ReLU Rectified Linear Unit.

SDS Spoken Dialog System.

SLU Spoken Language Understanding.

TD Temporal Difference.

TTS Text to Speech.

1 Introduction

“Hey Siri, ...”

Due the great success of automatic speech recognition systems (Hinton et al., 2012; Xiong et al., 2018), Spoken Dialog Systems (SDSs) have become an important research topic in academia and industry alike, giving rise to natural language based human-machine interaction systems like digital assistants. With Apple’s Siri, Microsoft’s Cortana, Amazon’s Alexa, Samsung’s Bixby or the Google Assistant, many big technology companies grant their users the ability to ask questions, change room temperature, find songs and purchase products by talking to a SDS.

Understanding the user’s input is only one part of an interactive system: somehow the system has to construct a reply based on some dialog strategy. In a typical task-driven SDS, which assists a user for example in booking a restaurant matching several criteria like price-range and food type or booking flights (Bobrow et al., 1977; Wen et al., 2016b), this is the responsibility of the *dialog policy*.

Depending on the task’s ontology, a mapping between all possible user inputs and system outputs might become extremely complex. Increasingly so, if the policy should obey some reasonable concepts like producing results matching the user’s criteria in as few dialog turns as possible. To complicate things even more, the input to such a system is generally noisy: imagine for example talking to your phone in a crowded and loud environment like a subway. This might cause the automatic speech recognition system to misunderstand the user, as might further factors like language ambiguity. Crafting dialog policies by hand is therefore time-consuming and error-prone and resulting policies are neither guaranteed to be efficient nor to be of high quality.

Nowadays, advancements in machine learning and especially deep reinforcement learning enable automatic learning of dialog policies. Aside from the benefit of reduced human effort, policies may also be trained to optimize

criteria like minimal number of required dialog turns to satisfy the user’s request. Deep reinforcement learning has been proven to be successful with deep Q-Networks (Mnih et al., 2013) using a neural network to approximate an action-value function and has been widely used in the context of dialog policy learning (Su et al., 2015; Dhingra et al., 2016; Casanueva et al., 2017). However, according to a recent comparison in the domain of dialog policy learning (Casanueva et al., 2017), it performed worse than other reinforcement learning methods such as Gaussian Processes (Gašić et al., 2010) in almost all testing conditions.

Recently, several advances in the area of deep reinforcement learning such as dueling network architectures (Wang et al., 2015) or distributional reinforcement learning (Bellemare et al., 2017) and their combination (Hessel et al., 2017) have been shown to be promising for further improvements of deep reinforcement learning agents in testing environments like Atari 2600 (Bellemare et al., 2013), which consists of several classic computer games. However, it is still unclear whether these methods could advance the dialog policies to outperform other methods and if they could reach the performance of handcrafted policies.

After an introduction to function approximation and reinforcement learning, the fusion of both is discussed in the form of deep Q-learning. Promising extensions to the basic deep Q-learning algorithm are presented and combined into one agent. The agent’s performance is evaluated on multiple dialog ontologies and varying testing conditions like input channel noise level and user friendliness, with results on par with handcrafted dialog strategies. It is compared to existing benchmarks of other reinforcement learning methods, proving its state-of-the-art performance and showing the potential of combining advanced deep reinforcement learning techniques in a meaningful context. Furthermore, the influence of each extension is tested and discussed in an ablation study, providing insight into the question which methods are essential for performance gain in the domain of dialog policy learning.

2 Background

In the following section, an overview of Spoken Dialog Systems is given, followed by an introduction to reinforcement learning and how the latter may be applied to solve the former.

2.1 Spoken Dialog Systems

SDSs allow human-machine interaction through spoken language. A SDS can be designed for generic purposes like small-talk (Weizenbaum, 1966) or for specific tasks such as finding restaurants (Bobrow et al., 1977) or booking flights (Wen et al., 2016b). In the following, only task-driven SDSs are considered. Therefore, the term SDS is used synonymously for a task-driven SDS.

Task-driven dialog systems try to assist a user achieving a certain goal. For example, a restaurant search system might help the user to identify candidates from a database matching certain user-specifiable criteria, like type of food, part of town and price-range. Depending on the task’s ontology, the amount of supported criteria and their number of known values may vary. Deciding how to respond to reach this goal given some user input is the responsibility of the dialog policy.

Figure 1 shows the architecture of a modular dialog system as described by Williams et al. (2016): A user talks to the system. Their spoken utterances are converted to text by an Automatic Speech Recognition (ASR) and then processed in a Spoken Language Understanding (SLU) unit which tries to extract task-relevant information and passes it to the Dialog State Tracker (DST). The DST accumulates the dialog history by storing and updating information from the SLU so that it can remember what information was already provided by the user. Based on this dialog state, the dialog policy decides on the next system action. More details on dialog policies are provided in the following section 2.1.1. Finally, a Natural Language Generator

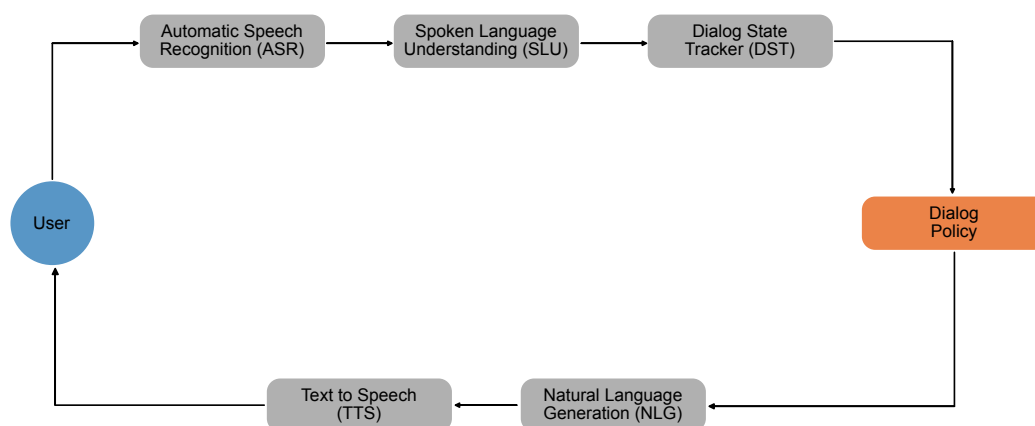


Figure 1: Schematic of a modular dialog system. Adapted from Williams et al. (2016).

(NLG) converts the system action in a human-readable representation and a Text to Speech (TTS) module transforms it into spoken language. This procedure, from user input to system output, is a dialog turn.

2.1.1 Dialog Policies

Dialog state here encompasses a list of all possible values for an ontology, coupled with a probability describing how sure the SDS is that a value has been mentioned by the user since the dialog’s beginning. Considering a dialog policy as a mapping between dialog state and the action the system can take to respond with, it becomes clear that such a mapping may become quite large. Further complications are caused by the possibility of noise in the input channel and language ambiguity. These additional factors could lead to an inaccurate estimation of the true dialog state, requiring the system to be able to explicitly ask the user for confirmation or to repeat some information and also to be able to recover from false assumptions. On top of that, a task-driven SDS is expected to behave reasonably and efficiently. That is, it should only take actions that are both sensible and necessary. For example, if a user informs the system that they want to eat French food, the SDS

should not ask about food type again and only propose French restaurants in the following interaction, except for the case where there are no matching results. In order to be efficient, questions asked by the SDS should reduce the size of possible matches or clear uncertainties. Accomplishing all of this involves planning a dialog strategy. Reinforcement learning, a technique in the area of machine learning, is designed to solve such planning problems.

2.2 Reinforcement Learning

In contrast to many other machine learning methods, reinforcement learning does not require collecting a large labeled corpus of training data in advance. Instead, it pursues a more dynamic approach by learning through interaction, thereby generating its own training data on the fly and actively steering the data generation process. The next sections explaining basic concepts of reinforcement learning are based on Sutton and Barto (1998).

2.2.1 Agent-Environment-Interaction

As visualized in figure 2, the learning unit, called agent, actively changes its environment by choosing to perform an action a from a set of possible actions, A . It perceives its environment as a state s , often provided by an artificial environment like a simulation. After an action was executed, the environment changes accordingly and tells the agent its new state s' . Additionally, the environment provides feedback about the goodness of the chosen action with respect to the given state by emitting a reward signal r to the agent.

An agent continues to act, thereby transitioning from state to state, until it reaches a terminal state. There are also non-terminal settings possible, but they are of no concern here because the maximum dialog turns are limited by a fixed number. All of the agent's experiences between starting and reaching

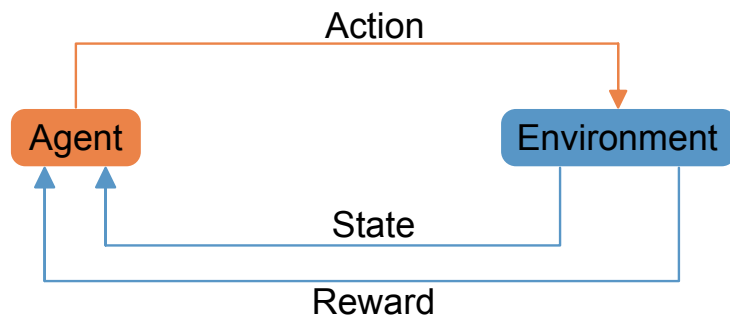


Figure 2: Interaction between agent and environment. Adapted from Sutton and Barto (1998).

a terminal state are called an *episode*. The list of visited states together with the encountered rewards and chosen actions in an episode is called *trajectory*.

However, to be able to model real-world scenarios, it is important to take uncertainty into account. For example, a person in danger of drowning but speaking with a lisp might be understood as shouting “I’m thinking!” instead of “I’m *sinking!*” by an automated emergency dispatcher. Mathematically, this is represented by introducing a transition probability function P which tells the probability to end up in another state s' if action a is performed in state s .

The goal of the agent is then to maximize the expected reward by learning to take actions which will result in the highest accumulated reward possible per episode. The behavior of the agent, meaning its action selection dependent on the state, is called policy and denoted π . Reinforcement learning is concerned with learning policies optimal with respect to the expected reward.

One of the key characteristics of reinforcement learning is the so-called exploration versus exploitation dilemma: If the agent wants to learn better actions, meaning actions which ultimately result in a higher accumulated reward, it has to try out new actions. But these are not guaranteed to lead to a higher return than the reward the agent could have gotten if it had exploited its current knowledge and followed the actions already known to

return good rewards. So this is the dilemma an agent is faced with when deciding which action to take next: either try random actions, speculating about getting higher rewards but with the risk of scoring worse, or achieving a possibly lower but secure reward by acting according to its current knowledge.

2.2.2 Markov Decision Process

Formalizing the above, a Markov Decision Process (MDP) is given by the tuple

$$(1) \quad (S, A, R, T, \gamma)$$

where S is the set of possible states, A the set of possible actions, R the reward signal defined by $R : S \times S \times A \rightarrow \mathbb{R}$ and T the state transition function with the mapping $T : S \times S \times A \rightarrow [0, 1]$ which has to satisfy the *Markov property* presented in equation 2.

$$(2) \quad \mathbb{P}(S_t = s_t | S_{t-1} = s_{t-1}, \dots, S_0 = s_0) = \mathbb{P}(S_t = s_t | S_{t-1} = s_{t-1})$$

It states that the probability of ending up in a state s_t is conditionally independent of the history of visited states s_{t-2}, \dots, s_0 and only influenced by the immediate predecessor state s_{t-1} . The state transition function obeying the Markov property can be defined as

$$(3) \quad T(s', s, a) = \mathbb{P}(s' | s, a)$$

to that T returns the probability of ending up in state s' when executing action a in state s . γ is a discount factor which dampens rewards the stronger the more they lie in the future.

2.2.3 Value functions

There are two general types of reinforcement learning problems (Sutton and Barto, 1998): If the transition and reward functions are known, the only

remaining problem is to plan the best route through the MDP. This problem class is called model-based learning, since the model of the environment is already known. Model-free learning in contrast has no initial knowledge about the transition function and has to learn it while also learning to find efficient paths. In general, it can be difficult to obtain these functions manually, for example the environment may be complex, difficult to measure or it might even change over time. In the following, only model-free learning methods are discussed.

Since the agent has to explore its environment in the model-free learning case, there are two possible ways the agent can handle its exploration: either the agent follows the same policy that it updates during the exploration, which is called *on-policy* learning, or the agent uses a different policy for exploration than the policy it uses for evaluation. This approach is called *off-policy* learning.

Knowing the transition and reward function is still not sufficient to produce a policy which maximizes the expected reward since rewards may be delayed and given only at the end of an episode. A policy which would traverse states by always choosing the action yielding the highest immediate reward would be very short-sighted and not necessarily lead to the best possible outcome, meaning the accumulated reward over a trajectory could become much higher by acting differently. Figure 3 illustrates this problem with a simple MDP where the initial state is s_0 with two possible actions a_1, a_2 that lead to their successor states s_1, s_2 with probability 1, meaning $P(s_1|s_0, a_1) = 1$ and $P(s_2|s_0, a_2) = 1$. Let action a_1 leading to state s_1 return a reward of $r_1 := -1$ and action a_2 leading to state s_2 a reward of $r_2 := +1$. State s_2 is followed by a state s'_2 to which the agent would be guaranteed to transition afterwards: $P(s'_2|s_2, a \in A) = 1$ and the rewards would yield $r'_2 := -100$. State s_1 has only one possible successor state s'_1 by letting $P(s'_1|s_1, a \in A) = 1$. If $\forall a \in A : R(s_1, a) > -97$, even though s_2 seems to be the better immediate choice because $r_2 > r_1$, all outcomes following a_1 are better than a_2 since the accumulated reward $R_1 := r_1 + r'_1 > -98$ is higher

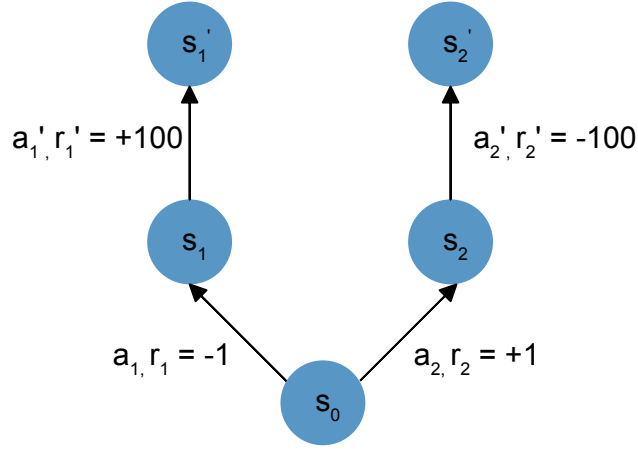


Figure 3: Simple MDP illustrating the necessity of planning.

than the accumulated reward $R_2 := r_2 + r_2' = -99$. As this example already indicates, looking at the accumulated rewards

$$(4) \quad R_t := \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

is a good choice for choosing the next state when planning more than one step ahead.

This leads to the idea of *value functions*, a base for many reinforcement learning algorithms. Since the transitions in an MDP might be probabilistic in contrast to the aforementioned example, a state value can not be calculated by simply accumulating rewards. Taking this into account, equation 5 defines the value function as the expected cumulative reward when starting in a state s and following policy π , where $\pi(a|s)$ is the probability of choosing action a in state s , from thereon.

$$(5) \quad v^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] \quad \forall s \in S$$

It can also be written in a recursive form (see equation 6) which is called the

Bellman equation (Bellman, 1957).

$$\begin{aligned}
 v^\pi(s) &= \mathbb{E}_\pi[R_t | S_t = s] \\
 (6) \quad &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s] \\
 &= \sum_a \pi(a|s) \sum_{s', a'} \mathbb{P}(s', r | s, a) [r + \gamma v^\pi(s')] \quad \forall s \in S
 \end{aligned}$$

Iterating equation 6 repeatedly with respect to a policy π converges to a fixed point v_π^* . An optimal policy π is found if there is no other policy π' for which the values $v_{\pi'}^*$ are greater than the ones of v_π^* .

2.2.4 Temporal Difference Algorithms

The reinforcement learning methods described in the following are instances of Temporal Difference (TD) algorithms (Sutton, 1988). Temporal difference algorithms are a class of model-free and bootstrapping reinforcement learning methods. Bootstrapping in this context means that the value functions are not updated after an episode has finished and the final reward along the corresponding trajectory is known but instead they can learn after each step by taking the estimated value function as a guess for the real accumulated reward.

The term TD-error is used in the following to describe the difference between the obtained (target) and predicted reward.

2.2.5 Q-Learning

While the value function describes how good a state s is overall, acting optimally with respect to v requires to look at the successor states' values reachable from s . The value function is the expected cumulative reward for one state s and therefore only dependent on s . It could be made more fine-granular by calculating an expectation for each state-action pair (s, a) , evaluating the expected reward when taking action a from state s . Such a

function is called *state-action* function or *Q*-function (Watkins, 1989). Compared to the value function, it already performs what Watkins (1989) calls the *one-step look-ahead*: the kind of look-ahead required for acting optimally with respect to the value function by directly yielding the expected reward for each possible action.

Equation 7 states the formal definition for the *Q*-value function analogous to the previously discussed value function.

$$\begin{aligned}
 (7) \quad q_*^\pi(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, a_t = a] \\
 &= \sum_{s', a'} \mathbb{P}(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')]
 \end{aligned}$$

The corresponding algorithm for calculating an optimal action-value function is called *Q-learning* which performs the update

$$(8) \quad Q(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

for some small update factor $\alpha \in \mathbb{R}$ and transition (s, a, r, s') .

Watkins (1989) proved that one-step *Q-learning* converges to an optimal action-value function. It is an off-policy learning method because it learns state-action values independent of the policy followed and it is also model-free because the transition probabilities need not be known.

2.3 Dialog Policies as Markov Decision Problems

Modelling dialog between a human and a machine as MDP is a simplified view of the problem because it assumes that the user's true inner intent is observable, which is clearly not possible. What can be observed are the user's utterances. However, the ASR could still misunderstand the utterances and thereby the information the user intended to provide, prohibiting full observability of user intent. Hence, a partially observable MDP would be a more realistic model but also a far less tractable one. To integrate some kind

of uncertainty for the MDP state, the belief tracker augments the state with probabilities from the ASR.

The state space S of the MDP is a vector over the probabilities of being mentioned for each slot-value pair concatenated with a vector consisting of a binary entry for each requestable slot, indicating if said slot was requested by the user. Also, information of the last system actions are encoded.

Representing the action space A follows a similar approach. For every possible system action like informing about food type, informing about the pricerange, asking for the food type or the pricerange, or generic actions like greeting, there is an action in the action space of the MDP.

Rewards are simply -1 per turn to motivate the agent to learn short dialogs and +20 if the user was provided with an answer matching their criteria, making successful dialogs more attractive to the agent than unsuccessful ones.

Maximum turn count is limited to 25 user-system interactions per dialog.

2.4 Neural Networks

From a high level perspective, machine learning algorithms can be characterized by three ingredients: The model which is used to represent the hypothesis, the objective or loss function and the optimizer used to minimize said loss function under the given model (Domingos, 2012).

In this thesis, the models are neural feed-forward networks. The following description of neural networks is based on Goodfellow et al. (2016), who define a neural feed-forward network as an acyclic directed graph with an arbitrary amount of layered neurons. Each input signal is connected to each neuron in the first layer and between layers each neuron is connected to all neurons of the following layer, as demonstrated in figure 4. Neurons in the same layer are not connected. A linear network layer h with k -dimensional input x and

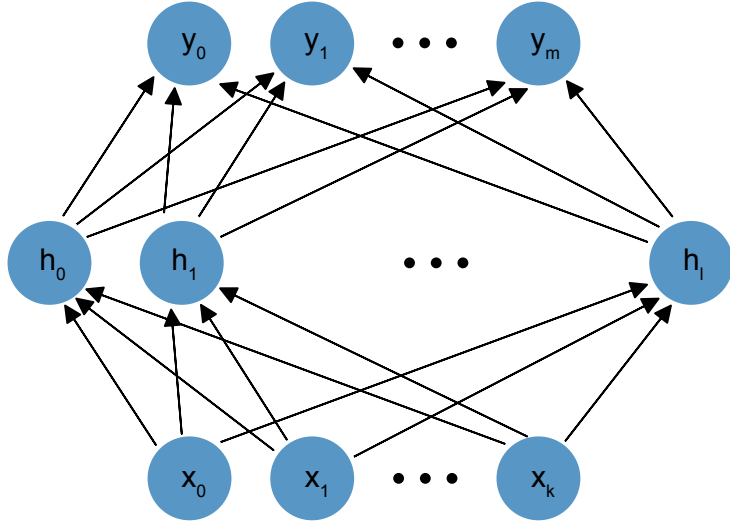


Figure 4: Example feed-forward network with input vector x , one hidden layer h and output layer y . Each arrow represents one trainable weight. Adapted from Goodfellow et al. (2016).

l neurons calculates the function

$$(9) \quad f_\phi(x) = W_l x + b_l$$

where W_l is a trainable $\mathbb{R}^{k \times l}$ matrix and b_l a trainable \mathbb{R}^l bias vector. ϕ denotes the combined layer weights $\phi = (W_l, b_l)$.

Between two layers, each signal has to pass a non-linear activation function. Popular choices are sigmoid functions or Rectified Linear Units (ReLUs). By applying for example the sigmoid function element-wise to equation 9, one obtains

$$(10) \quad h_\phi(x) = \sigma(f_\phi(x)) = \sigma(W_l x + b_l)$$

as the final output of layer h . Networks stacking multiple hidden layers, meaning the output of one layer is used as input for a following layer, are often referred to as deep networks. Training such multi-layered networks is called deep learning.

Learning to approximate a certain function is then done via a so-called supervised training process. Given a known label y for the input x , one propagates x through the neural network, yielding the current approximation \hat{y} . The loss function \mathcal{L} then measures the difference between the real output and the approximate output and penalizes it, for example using the squared Euclidean norm:

$$(11) \quad \mathcal{L} = \|y - \hat{y}\|_2^2$$

Weights of the neural networks are then updated using a back-propagation algorithm (Rumelhart et al., 1986) which propagates the gradient from the loss backwards through the network to the inputs. An optimizer then adapts the weights by moving them a typically small amount into the direction of the corresponding gradient.

Hornik et al. (1989) showed that neural multilayer feed-forward networks can approximate any Borel-measurable function mapping between finite dimensional spaces to an arbitrary degree of accuracy given a sufficient amount of neurons. Therefore, they represent a class of universal approximators.

3 Related Work

In this chapter, current methods for dialog policy learning and a specific reinforcement learning method called Deep Q-Networks (DQN) along with several extensions are presented.

3.1 Dialog Policy Learning

Early dialog policy learning practices relied on tabular reinforcement learning methods as described in the previous chapter. For example, Walker (2000) trained an Email voice interaction system with real users. More recent methods rely on deep reinforcement learning algorithms. But one problem of these

approaches is that they require large amounts of training data, rendering training with real users infeasible. For this purpose, user simulators like the one from Schatzmann et al. (2007) were created. With the possibility of producing arbitrary amounts of training dialogs by being able to communicate with a simulated user, deep reinforcement learning methods became practical for learning dialog policies. Examples of the quality of automatically learned dialog systems can be seen in the benchmark by Casanueva et al. (2017), which provides an overview over multiple reinforcement learning agents, ranging from Gaussian Process SARSA (GP-SARSA) (Gasic and Young, 2014) to Adversarial Advantage Actor-Critic Models (A2C) (Peng et al., 2018).

Recent developments even allow training a whole SDS in end-to-end fashion, meaning all components like the SLU are trained simultaneously with the dialog policy (Wen et al., 2016a).

3.2 Deep Reinforcement Learning

Since neural networks are universal function approximators, they may also be used to approximate Q-functions. Benefits over tabular schemes are manifold. For one, inputs to the neural network need no longer be discretized. This especially benefits the dialog policy setting here because input states contain continuous probabilities. In comparison to tabular methods, neural networks may also generalize to unseen states and actions. Lastly, they may be able to learn better representations for the input states than a human could provide. However, there are some changes necessary to the basic Q-learning algorithm to make it work well with neural networks.

One major difference is the integration of a so-called experience replay buffer (Lin, 1993). When training the neural network, experience tuples of the form (s, a, r, s') are stored in said buffer instead of being directly used for training. During training, they are sampled randomly from this buffer which decorrelates the agent’s experiences which is a base assumption for

the convergence of many supervised learning algorithms. Another benefit is that since exploration involves random decisions many transitions may be encountered only once and neural networks typically require to see samples multiple times to permanently learn from them. Sampling from a buffer therefore increases the chance for such transitions to be seen multiple times when training the neural network using mini-batches.

Since the development of the first neural Q-learning algorithms, like the ones by Lin (1993) and Riedmiller (2005), many improvements and extensions were proposed which are introduced in the following sections.

3.2.1 Deep Q-Networks

One of the most popular basic model-free off-policy learning algorithms with neural network function approximation is the DQN algorithm presented by Mnih et al. (2013). The Q-function is represented by a neural network with parameters ϕ . The tabular update rule of standard Q-learning given in equation 8 is no longer applicable in the function approximation setting. In order to optimize the network weights, a quadratic loss function on the TD-error is calculated for each step i with transition (s, a, r, s') as presented in equation 12. This loss function can then be minimized using a suitable gradient descent method.

$$(12) \quad L_i(\phi_i) = (r + \gamma \max_{a'} Q(s', a', \phi_i) - Q(s, a, \phi_i))^2$$

3.2.2 Prioritized Experience Replay

As already mentioned in the context of the deep Q-learning algorithm with experience replay (Mnih et al., 2013), uniform sampling from an experience replay buffer might not be the best strategy because important experiences from which the agent might learn a lot are sampled with the same probability as experiences which do not teach the agent anything new.

Prioritized Experience Replay by Schaul et al. (2015) improves on this concept by assigning each experience a priority. Whenever the agent follows a new transition, the corresponding experience is inserted with maximum probability into the buffer, thereby introducing a bias towards sampling newer experiences. This is reasonable because newer transitions were obtained by a newer policy and therefore contain new information.

The priorities p_t are chosen as the absolute of the TD-error

$$(13) \quad p_t = |\delta_t|$$

for transition t with δ_t being the TD-error of transition t . This follows the intuition that a transition with a high TD-error means that the agent can learn much from this transition because the current estimate is still far from the target. Since we train for a relatively small amount of episodes and there are so many different ways a dialog may continue, it is expected that this concept should prove beneficial here.

Probabilities for drawing sample t are obtained by converting the priorities to a valid probability distribution

$$(14) \quad \mathbb{P}(t) = \frac{p_t^\alpha}{\sum_k p_k^\alpha}$$

where the exponent α steers the prioritization. Setting $\alpha := 0$ for example gives the probabilities

$$(15) \quad \mathbb{P}(t) = \frac{p_t^0}{\sum_k p_k^0} = \frac{1}{\sum_k 1} = \frac{1}{B}$$

where B is the buffer size. This is exactly the uniform sampling case. Evidently, the bigger α is chosen the less uniform the distribution becomes.

But sampling non-uniformly introduces bias in the calculation of the expected reward. To obtain an unbiased estimate, the TD-error for sampled experience t has to be corrected by the weights from equation 16.

$$(16) \quad w_t = \left(\frac{1}{B} \cdot \frac{1}{P(t)}\right)^\beta$$

If $\beta = 1$, the correction removes the bias from the estimate. Schaul et al. (2015) also mention the possibility of annealing β linearly to 1 and to normalize the weights by the maximum weight.

3.2.3 Double DQN

As proven by Van den Steen (2004) and Smith and Winkler (2006) agents acting greedily on their estimated values produce a biased estimate of the expected value because usage of the maximization operator can lead to over-estimation of the expectation.

As Hasselt (2010) claims, double Q-learning should prove beneficial in noisy environments. Including this technique in the dialog policy learning is interesting because this domain is inherently noisy and the user simulator used for training the reinforcement learning agents induces noise in many tasks in order to make them more realistic. Because dialog systems are inherently noisy due to input channel noise, this extension could improve the stability and performance of the reinforcement learning agent.

Van Hasselt et al. (2016) introduced the Double DQN algorithm which mitigates this problem. For Double DQN learning, the Q-network $Q(s, a, \phi)$ where ϕ are the network parameters is called the online network, is updated during training. A second Q-network $Q(s, a, \phi^-)$ called target network with parameters ϕ^- is introduced which is not trained on but held constant. From time to time, the online network’s parameters are copied to the target network. Compared to the basic deep Q-learning algorithm, the calculation of the TD-error changes to make use of this secondary neural network. While the next best action is still chosen greedily with respect to the online network, the corresponding action-state value is estimated by the older target network instead of the online network:

$$(17) \quad \delta = r + \gamma Q(s', \underset{a'}{\operatorname{argmax}} Q(s', a', \phi), \phi^-) - Q(s, a, \phi)$$

for one transition (s, a, r, s') .

3.2.4 Multi-Step Targets

Sutton (1988) proposed to expand the one-step TD-methods to multi-step views in order to speed up the learning process. The idea is that the agent may benefit from knowledge of future states which has not yet been propagated to the current state's immediate successor states which is achieved by looking at the accumulated reward for the next n steps. Because n is chosen constant, this method is somewhere between one-step TD-methods and a full Monte-Carlo scheme which would look at the accumulated reward of a complete episode instead of relying on a finite horizon. Equation 18 displays the accumulated n -step reward for the target in the corresponding multi-step TD-error between state s and s_{t+n} in equation 19. Because we train for relatively few episodes, a bootstrapping mechanism like multi-step targets could benefit the agents.

$$(18) \quad R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}$$

$$(19) \quad \delta = R_t^{(n)} + \gamma^k \max_{a'} Q(s_{t+n}, a') - Q(s_t, a)$$

3.2.5 Noisy Networks for Exploration

Many reinforcement learning algorithms explore using the already described ϵ -greedy exploration method. But this approach has the disadvantage that it requires additional hyperparameters because the curve of the ϵ -decay has to be chosen and tuned manually and the agents introduced in the following sections already have a large amount of hyperparameters, making it all the more difficult and time-consuming to optimize them.

Noisy Linear Layers (Fortunato et al., 2017) on the other hand do not require manual parameters except for an initial noise level factor. The idea is to let the network find out the required amount of exploration by itself and adjust it during the training process. To achieve this, the calculation of a linear layer mapping from $\mathbb{R}^k \rightarrow \mathbb{R}^l$ (see chapter 2.4, equation 9) is changed to

$$(20) \quad f_\phi(x) = (W_l + W_l^\epsilon \odot \epsilon^w)x + b_l + b_l^\epsilon \odot \epsilon^b$$

with the weights W_l and b_l representing the standard weight matrix and bias vector. ϵ^w and ϵ^b are random noise variables and W_l^ϵ and b_l^ϵ noise weight matrix and noise bias weights. Again, ϕ represents all layer weights $\phi = (W_l, b_l, W_l^\epsilon, b_l^\epsilon)$

By multiplying the random variables with trainable weights, the network can learn to increase or dampen the noise as needed. For example, a network with noisy layers could learn to increase the noise level at first to increase exploration in the beginning and later to dampen the noise again to end exploration when no further policy improvement is attained.

Fortunato et al. (2017) present two possibilities for the choice of the noise random variables: independent Gaussian noise, where every entry of ϵ^w and ϵ^b is sampled independently from a unit Gaussian distribution, or factorised Gaussian noise with samples ϵ which we use here because it requires less computation effort by choosing

$$(21) \quad \epsilon_{i,j}^w = f(\epsilon_i)f(\epsilon_j) \quad 1 \leq i \leq k, 1 \leq j \leq l$$

and

$$(22) \quad \epsilon_j^b = f(\epsilon_j) \quad 1 \leq k \leq l$$

with

$$(23) \quad scale(\epsilon) = sgn(\epsilon)\sqrt{|\epsilon|}$$

as a scaling function.

3.2.6 Dueling network architecture

The action-value function was presented as a more fine-granular estimation of utility, dependent not only on the state like the value function but also on the action choice in this state. But for states where the choice of action does not matter, knowing the value function is a sufficient level of granularity since the output of the action-value function would be the same for each possible action. If some actions matter, the advantage function could at least be of small complexity, meaning the outputs should produce similar values for many actions. In the dialog policy setting, one would expect such states to occur, for example at the beginning of a dialog, the system should try to gather information. Which information exactly should not matter too much if the ontology requires all slots to be filled to be able to find a good match to the user’s input. Other actions should be less important in this case. Also, at the end of a dialog where the system already presented a result, further information gathering should be equally unlikely for all slots since they are already filled - only dialog ending actions or coming up with alternatives should be suitable and differently rated actions.

Based on this insight, Wang et al. (2015) constructed a third utility estimation function by connecting the value function and the action-value function via the equation

$$(24) \quad A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

which they called advantage function. Instead of measuring the expected reward by executing action a in state s like the standard action-value function, the advantage function measures the difference of utility which action a provides compared to the base utility of the state.

Rewriting equation 24 as

$$(25) \quad Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$$

one obtains the action-value function expressed by the value function and the advantage function. States where choice of action is not of importance

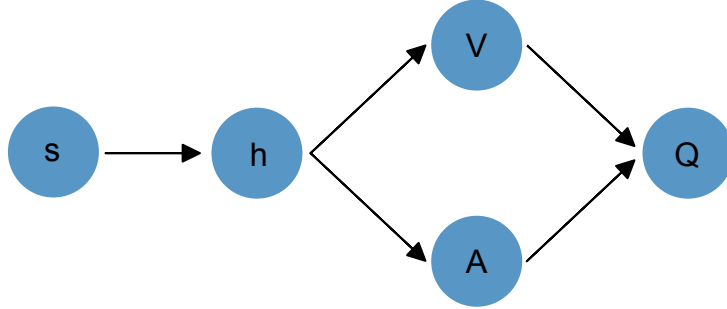


Figure 5: Schematic of a dueling network. The input state s is processed by a shared layer h from which the calculation is split into independent layers V and A , calculating the value and advantage functions. In the output Q , both functions are merged again to form the Q-value. Adapted from Wang et al. (2015).

now may learn just the value function for this state since $A^\pi(s, a) = 0$ for such states.

Equation 24 is realized by a feed-forward neural net where both the value and the advantage function share an encoding layer h for the state with parameters ϕ_s , which calculates the function $h_{\phi_s}(s)$ as in equation 10.

From there, advantage and value function are calculated with separate layers with parameters ϕ_α and ϕ_β . They are then joined again according to equation 26 to calculate the action-value function as output. The architecture of the corresponding network implementing this functionality is shown schematically in figure 5.

$$(26) \quad Q^\pi(s, a, \phi) = V^\pi(h_{\phi_s}(s), \phi_\beta) + A^\pi(h_{\phi_s}(s), a, \phi_\alpha)$$

However, equation 25 clearly does not allow for unique recovery of both $V(s, a)$ and $A(s, a)$ given $Q(s, a)$ since it is under-determined. To alleviate this problem, Wang et al. (2015) propose to subtract the maximum value of the advantage function so that $Q(s, a^*) = V(s)$ when following the best action $a^* = \operatorname{argmax}_{a'} Q(s, a')$. In practice, they found that replacing the

maximization with an average over all actions like

$$(27) \quad Q(s, a, \phi) = V(h_{\phi_s}(s), \phi_\beta) + A(h_{\phi_s}(s), a, \phi_\alpha) - \frac{1}{|A|} \sum_{a'} A(h_{\phi_s}(s), a', \phi_\alpha)$$

leads to more stable learning.

3.2.7 Distributional DQN

When an agent is following a trajectory, rewards along that trajectory may be random. This is partly due to the generally non-deterministic nature of state transitions in MDPs as described in chapter 2.2.2. Randomness might further be introduced by acting according to a non-deterministic policy like the ϵ -greedy policy.

All previously described methods are based on the action-value function mapping $Q(s, a)$ from state-action pairs to the expected reward for taking action a in state s , thereby discarding information about distribution of the rewards. One of the most recent additions to deep Q-learning concerned with learning the full distribution of rewards $Z(s, a)$ is the Quantile Regression DQN (QR-DQN) algorithm presented by Dabney et al. (2017). Again, an inherently noisy environment like real-world dialog could benefit from better knowledge of the distribution of rewards.

This return distribution Z for state s and action a is modelled by a Cumulative Distribution Function (CDF) F defined as $F(Z(s, a)) = \tau$, meaning that the probability of obtaining a random reward r smaller or equal to $Z(s, a)$ is τ . Or, looking at the inverse CDF, one obtains the quantile distribution $F^{-1}(\tau) = Z(s, a)$.

In order to minimize the difference between the target and estimated quantile distributions Y and Y' , the p-Wasserstein metric between both distributions

$$(28) \quad W_p(Y, Y') = \left(\int_0^1 |F_Y^{-1}(\omega) - F_{Y'}^{-1}(\omega)|^p d\omega \right)^{\frac{1}{p}}$$

is minimized.

Constructing the approximation requires discretisation of the probabilities first. Choosing N uniformly spaced probabilities q yields

$$(29) \quad q_i = \frac{1}{N} \quad \forall i : 1 \leq i \leq N$$

and results in

$$(30) \quad \tau_i = \frac{i}{N} \quad \forall i : 1 \leq i \leq N$$

which are the discrete cumulative probabilities τ .

A parametric version Z_θ of the estimated distribution on a discrete support with parametric model θ is constructed as

$$(31) \quad Z_\theta(x, a) := \frac{1}{N} \sum_{i=1}^N \delta_{\theta_i(x, a)}$$

by the aid of N Diracs δ_z .

Calculating the W_1 -metric between a continuous distribution Z and the discrete parametric distribution Z_θ can be done via equation 32.

$$(32) \quad W_1(Z, Z_\theta) = \sum_{i=1}^N \int_{\tau_{i-1}}^{\tau_i} |F_Z^{-1}(\omega) - \theta_i| d\omega$$

The discrete distribution Z_θ as a projection from Z onto the space of discretized parametric distributions with the minimal error in the W_1 -metric is then defined as

$$(33) \quad \Pi_{W_1} Z := \underset{Z_\theta}{\operatorname{argmin}} W_1(Z, Z_\theta)$$

where Π is the projection operator.

The θ_i minimizing $W_1(Z, Z_\theta)$ are found on the midpoints between two consecutive cumulative probabilities τ_i and τ_{i+1}

$$(34) \quad \hat{\tau}_i := \frac{\tau_i + \tau_{i+1}}{2}$$

yielding $\theta_i = F_Z^{-1}(\hat{\tau}_i)$.

Obtaining an unbiased quantile function approximation for this parametrisation can be achieved by minimizing the quantile regression loss

$$(35) \quad \mathcal{L}_{QR}^\tau(\theta) = \mathbb{E}_{\hat{Z} \sim Z}[\rho_\tau(\hat{Z} - \theta)]$$

with

$$(36) \quad \rho_\tau(u) = u(\tau - \mathbb{I}_{u < 0})$$

Given the discrete distribution Z_θ with N and the minimizing $\hat{\tau}$, the final loss is given by equation 37.

$$(37) \quad \mathcal{L}(\theta) = \sum_{i=1}^N \mathbb{E}[\rho_{\hat{\tau}_i}(\hat{Z} - \theta_i)]$$

Since Q-values are the expectation of the Z-distribution and the probabilities are evenly spaced, converting the distribution to a single Q-value which can be used for example to select actions can be done via equation 38.

$$(38) \quad Q(s, a) = \sum_{i=1}^N q_i \theta_i(s, a)$$

The TD-error for transition (s, a, r, s') is calculated pairwise according to

$$(39) \quad \delta_{ij} = r + \gamma \theta_j(x', a^*) - \theta_i(x, a) \quad \forall i, j \in \mathbb{N} : 1 \leq i, j \leq N$$

where a^*

$$(40) \quad a^* = \underset{a'}{\operatorname{argmax}} Q(s', a')$$

is the greedily chosen action.

3.2.8 Rainbow

Hessel et al. (2017) constructed a deep reinforcement learning agent by starting with the DQN algorithm (Mnih et al., 2013) and extending it with prioritized experience replay, noisy linear layers for exploration, double Q-learning,

dueling network architectures, multi-step learning and distributional reinforcement learning and called it the *Rainbow* agent. Evaluation of the Rainbow agent was performed on the *Atari 2600* environment (Bellemare et al., 2013). At the time of publication this environment consisted of 50 emulated games chosen from the catalogue of available games for the Atari 2600 video game console. State information is provided in the form of game screens consisting of a 2D array of pixels with dimension 160×210 . Every pixel’s colour is encoded with 7 bit. Rewards are given by game score difference between two consecutive game screens.

In the experiments conducted with the Rainbow agent on the Atari 2600 benchmark, including 7 addition games not available in the original Atari environment, the Rainbow agent was found to perform substantially better than the vanilla DQN algorithm and all agents leaving out one of the extensions (Hessel et al., 2017).

4 Resources

4.1 OpenAI Gym

The OpenAI Gym toolkit (Brockman et al., 2016) is a reinforcement learning toolkit providing several different environments. One of the included environments is the cart-pole environment where a cart may be steered by left or right drive commands. Attached to the top of the car is a pole which has to be balanced by moving the car. If the pole falls over or the car leaves a pre-defined area, the episode is terminated. Each action returns a reward of +1 which motivates the reinforcement learning agents to try to balance the pole as long as possible. The state describes the cart’s position and velocity together with the pole’s angle and velocity. It is initialized randomly from a certain range.

4.2 PyDial

The PyDial toolkit (Ultes et al., 2017) is an implementation of a modular SDS. Inputs from external speech clients, natural language text input and input on a semantic level by the integrated user simulator are supported. Processing the first two types of input is done by a semantic decoder module. Semantically decoded inputs or inputs from the user simulator are then fed to a belief tracking module that updates its belief of the current dialog state which maintains data like information on constraints the user has already mentioned. Due to the possibility of input channel noise, this belief is probabilistic, meaning a valid belief state could for example tell that a user mentioned *French food* with a probability of 80%.

A dialog policy module then decides on the answer the system should give based on the belief state, subject to the principle that it should help fulfill the user’s goals accurately and as fast as possible. How such a policy can be obtained via reinforcement learning is described in the next section.

After an action is chosen, a language generator module converts it to natural language output in text form and presents it to the user.

The ontologies used for the benchmarks in this paper together with their properties are listed in table 1.

Domain	#slots	#requests	#values
CR	3	9	268
SFR	6	11	636
LAP	11	21	257

Table 1: List of benchmark ontologies, the amount of slots the user can provide or request from the system as well as the total amount of values of each requestable slot Casanueva et al. (2017).

In Casanueva et al. (2017), six different environmental models were proposed, varying in user friendliness, simulated input channel noise and the

presence or absence of action masks, which, when enabled, simplify learning by masking some of the possible actions. An overview of all these environmental configurations and their assignment to tasks is given in table 2. Evaluation results in Casanueva et al. (2017) with several dialog policy types, e.g. a handcrafted policy and several learned policies serve as baseline in our experiments.

Task	Env. 1			Env. 2			Env. 3			Env. 4			Env. 5			Env. 6		
	T1.1	T1.2	T1.3	T2.1	T2.2	T2.3	T3.1	T3.2	T3.3	T4.1	T4.2	T4.3	T5.1	T5.2	T5.3	T6.1	T6.2	T6.3
Domain	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP	CR	SFR	LAP
SER	0%			0%			15%			15%			15%			30%		
Masks	On			Off			On			Off			On			On		
User	Standard			Standard			Standard			Standard			Unfriendly			Standard		

Table 2: Benchmarking tasks with task name, domain, semantic error rate (SER), action masking and user model Casanueva et al. (2017).

5 Methods

First, dialog management is cast into a MDP. Following the methodology of the Rainbow agent, a combined agent is then constructed to find optimal policies using the basic DQN algorithm and extensions described in chapter 3.2. After the presentation of the combined agent some alternative agents are proposed, each leaving out one of the extensions.

5.1 Combined agent

The fully combined agent extends the DQN algorithm with prioritized experience replay, double DQN, dueling network architecture and distributional learning method.

In contrast to the Rainbow agent, the multi-step method was dropped and instead of noisy linear layers, regular linear layers and ϵ -greedy exploration are used. While the first technique led to faster learning for some

tasks, both techniques harmed final agent performance in terms of evaluation rewards in most cases. The Rainbow agent relies on the Categorical DQN algorithm developed by Bellemare et al. (2017) whereas the combined agent here makes use of the more recent QR-DQN algorithm as described in chapter 3.2.7 which is able to minimize the Wasserstein-metric, no longer requires knowledge about value bounds and also frees from the restriction of a fixed value resolution.

Given a MDP (S, A, T, R, γ) and choosing a distribution over N quantiles, the dueling network θ defines the mapping $S \rightarrow \mathbb{R}^{|A| \times N}$ for estimating the distributional action-value function.

For the sake of familiarity, θ is described dependent on the state-action pair (s, a) instead of the state s only because the resulting equations bear more similarity to the formulae described in previous chapters this way. The actual implementation follows the state parametrisation.

Values for the quantiles are calculated according to the dueling network architecture with shared layer dimension \mathbb{R}^d by

$$(41) \quad \theta_i(s, a) = v_{\phi_\beta}^i(h_{\phi_s}(s)) + a_{\phi_\alpha}^i(h_{\phi_s}(s), a) - \frac{1}{|A|} \sum_{a'} a_{\phi_\alpha}^i(h_{\phi_s}(s), a')$$

$\forall i \in \mathbb{N} : 1 \leq i \leq N$ where $h_{\phi_s} : S \rightarrow \mathbb{R}^d$ represents the feed-forward function the shared network layers implement, $v_{\phi_\beta} : \mathbb{R}^d \rightarrow \mathbb{R}^N$ the feed-forward function of the value stream and $a_{\phi_\alpha} : \mathbb{R}^d \rightarrow \mathbb{R}^{|A| \times N}$ the feed-forward function of the advantage stream with the set of parameters $\phi = (\phi_s, \phi_\alpha, \phi_\beta)$.

Analogous to the double DQN, a target network θ^- with parameters $\phi^- = (\phi_s^-, \phi_\alpha^-, \phi_\beta^-)$ is introduced and held fix, meaning it is not trained on. Its weights are periodically updated by copying them from the online network.

Using both the online and the target network, the TD-error for a transition (s, a, r, s') is given by

$$(42) \quad \delta_{i,j} = r + \gamma \theta_j^-(s', a^*) - \theta_i(s, a)$$

where action a^* is chosen greedily according to the equation

$$(43) \quad a^* = \underset{a'}{\operatorname{argmax}} \sum_k q_k \theta_k(s', a')$$

with the probabilities

$$(44) \quad q_i = \frac{1}{N} \quad \forall i \in \mathbb{N} : 1 \leq i \leq N$$

. Cumulative probabilities τ_i are then given by

$$(45) \quad \tau_i = \frac{i}{N} \quad \forall i \in \mathbb{N} : 1 \leq i \leq N$$

The midpoint quantile targets $\hat{\tau}_i$ over which the loss is minimized can then be calculated as

$$(46) \quad \hat{\tau}_i = \frac{\tau_{i-1} + \tau_i}{2} \quad \forall i \in \mathbb{N} : 1 \leq i \leq N$$

Finally, the loss

$$(47) \quad \mathcal{L}(s, a, r, s') = \sum_{i=1}^N \mathbb{E}_j[\rho_{\hat{\tau}_i}^\kappa(\delta_{i,j})]$$

where ρ represents the smooth Quantile Huber Loss

$$(48) \quad \rho_\tau^\kappa(u) = |\tau - \mathbb{I}_{\{u < 0\}}| \mathcal{L}_h^\kappa(u)$$

and \mathcal{L}_h^κ the Huber Loss

$$(49) \quad \mathcal{L}_h^\kappa(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \leq \kappa \\ \kappa(|u| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases}$$

is minimized.

New transitions are added with maximum priority to the replay buffer. Sampling probabilities are chosen proportional to the loss to the power of $\alpha \in \mathbb{R}$ by transforming priorities p to probabilities P according to

$$(50) \quad \mathbb{P}(t) = \frac{p_t^\alpha}{\sum_{k=1}^B p_k^\alpha}$$

for a prioritized experience replay buffer of size B . To mitigate the bias introduced by the non-uniform sampling, the final loss for transition t is multiplied by the weighting

$$(51) \quad w_t = \left(\frac{1}{N} \cdot \frac{1}{P(t)}\right)^\beta$$

5.1.1 Combined agent without dueling network architecture

Compared to the combined agent, the agent without dueling network architecture is constructed exactly the same except for the architecture of the neural network which is replaced by a traditional feed-forward network directly calculating the mapping $f : S \rightarrow \mathbb{R}^{|A| \times N}$, simplifying the calculation of θ to

$$(52) \quad \theta_i(s, a) = f_{\phi_s}^i(s, a) \quad \forall i \in \mathbb{N} : 1 \leq i \leq N$$

5.1.2 Combined agent without distributional learning method

The agent without the distributional reinforcement learning method calculates the mapping $\theta : S \rightarrow \mathbb{R}^{|A|}$, meaning it calculates one Q-value per state-action pair instead of a distribution over rewards:

$$(53) \quad \theta(s, a) = v_{\phi_\beta}(h_{\phi_s}(s)) + a_{\phi_\alpha}(h_{\phi_s}(s), a) - \frac{1}{|A|} \sum_{a'} a_{\phi_\alpha}(h_{\phi_s}(s), a')$$

as in the normal dueling network algorithm. Dimensionality of the mappings v_{ϕ_β} and a_{ϕ_α} changes to $v_{\phi_\beta} : \mathbb{R}^d \rightarrow \mathbb{R}$ and $a_{\phi_\alpha} : \mathbb{R}^d \rightarrow \mathbb{R}^{|A|}$ accordingly.

It follows that the TD-error has to be changed from pairwise to the double DQN variant, dropping the indexing scheme and using the greedy action selection scheme:

$$(54) \quad \delta = r + \gamma \theta^-(s', \operatorname{argmax}_{a'} \theta(s', a')) - \theta(s, a)$$

.

Also, the loss has to be adapted to a non-distributional variant. Here, the Huber Loss \mathcal{L}_h is calculated on the TD-error.

5.1.3 Combined agent without double Q-learning

Target network θ^- is simply removed in the variant without double Q-learning, leading to

$$(55) \quad \delta_{i,j} = r + \gamma\theta_j(s', a^*) - \theta_i(s, a)$$

as TD-error calculation.

5.1.4 Combined agent without prioritized experience replay

The agent without prioritized experience replay is almost the same as the combined agent except for the sampling probabilities from the buffer which are always uniform in this variant. Importance weighting is therefore also removed.

5.1.5 Combined agent with multi-step targets

For the multi-step method, the target in the TD-error from equation 42 is replaced by a discounted reward accumulated over a trajectory $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \dots, s_{t+n}, a_{t+n}, r_{t+n})$ of n transitions:

$$(56) \quad \delta_{i,j} = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n \theta_j^-(s_{t+n}, a^*) - \theta_i(s_t, a_t)$$

5.1.6 Combined agent with noisy linear layers

Including noisy linear layers for exploration does not alter the equations given in this chapter - it just changes the network architecture and therefore the network parameters ϕ . Each linear layer is replaced with a noisy linear layer and the ϵ -greedy exploration scheme is omitted. Factorized noise for the online and the target networks is resampled after each action selection and also before each train call.

6 Results and Discussion

First, the choice of hyperparameters is explained which were used to obtain the following results which are then discussed to obtain some insights on agent performance depending on the combination of DQN extensions.

6.1 Hyperparameters

The hyperparameters were found by random search, followed by some manual fine-tuning. All neural network layers are fully connected linear layers with ReLUs as activation functions. In case of the dueling network architecture, the shared layer consists of 256 neurons, followed by two value layers, each with 300 neurons, and two advantage layers with 400 neurons per layer. Distributional agents use an atom count of 50. Where the dueling architecture is replaced by a standard architecture in the evaluation process, three layers of sizes 256, 700 and 700 are used to guarantee a fair comparison to the dueling case by providing the same model capacity. For prioritized replay, the prioritization exponent α is set to 0.525 and importance sampling exponent β to constant 0.4 without annealing. As Optimizer, we use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-4} . Exploration is performed ϵ -greedy with linear ϵ decay, starting at 0.3. Whenever an agent makes use of double Q-learning, it updates its target network after 6 dialogs. All agents use an experience replay buffer capacity of 16384 transitions, a discount factor $\gamma = 0.99$, mini-batch size 64 and the Huber Loss κ is set to 1. Following the PyDial benchmarking process, we leave all hyperparameters constant across all environments and dialog domains (Casanueva et al., 2017), thus also evaluating the generalization capabilities of the agents.

Results for the cart-pole testing environment were obtained with the hyperparameters in table 13.

6.2 Experiment setup

Each combined agent presented in chapter 5.1 is trained and tested on all 18 tasks listed in table 2 to assess its generalisation capabilities. For example, one would expect all agents to perform reasonably well on tasks without simulated input channel noise but their performance may alter significantly across different noise levels. For the same reason, hyperparameters are kept constant across agents and tasks.

In order to evaluate the performance of the agents in the context of dialog systems, they are compared to all of PyDial’s dialog management methods listed by Casanueva et al. (2017). To ensure a fair comparison, the training process of PyDial was followed, meaning each policy was obtained by training 4 iterations of 1000 dialogs each and evaluating on 500 test dialogs after each iteration.

Each experiment was repeated with five randomly chosen initializations for the random number generator, except for the agents with noisy linear layers for exploration (tables 9 and 10) and multi-step targets (tables 11 and 12), which were trained and evaluated on three random initializations only since it was already apparent after obtaining the first results that more seeds were unlikely to produce better results. Reported results are the performances on the last 500 test dialogs averaged across the outcomes of the five random seeds.

6.3 Results

The first row of tables 3, 4 show the results of the highest scoring policy from the PyDial benchmark Casanueva et al. (2017) to serve as baseline. Evaluations of the handcrafted policies follow in the last line. Even though PyDial comes with a DQN implementation, a custom version was implemented to have a fair baseline which shares the codebase and model parameters with the other policy models. The combined agent makes use of all the advanced

methods such as prioritized experience replay, double Q-learning, dueling network architecture and distributional reinforcement learning.

Task	best PyDial	DQN	combined	-distributional	-double	-prioritization	-dueling	handcrafted
T1.1	13.5 ¹	14.0	13.7	13.9	13.8	12.0	12.9	14.0
T1.2	11.7 ²	9.7	10.7	12.0	8.6	4.6	9.0	12.4
T1.3	10.5 ²	8.6	11.3	11.5	11.0	9.5	10.9	11.7
T2.1	12.2 ¹	11.4	13.0	13.2	12.8	13.3	11.7	14.0
T2.2	9.6 ¹	10.0	11.5	10.7	11.0	11.6	10.8	12.4
T2.3	7.3 ¹	4.0	10.2	9.6	10.1	9.2	9.8	11.7
T3.1	11.9 ³	12.5	13.0	12.8	13.1	13.1	12.8	11.0
T3.2	8.6 ²	8.6	9.4	10.1	9.3	9.0	5.7	9.0
T3.3	6.7 ²	7.3	8.7	8.8	7.7	7.9	7.2	8.7
T4.1	10.7 ³	8.3	11.4	11.2	11.7	11.7	10.9	11.0
T4.2	7.7 ³	9.3	10.4	9.5	8.9	9.0	6.1	9.0
T4.3	5.5 ³	2.2	9.4	9.9	8.5	9.1	6.5	8.7
T5.1	10.5 ²	10.5	11.9	11.5	11.9	12.0	11.7	9.7
T5.2	4.5 ²	5.3	6.4	7.9	7.5	6.7	5.6	6.4
T5.3	4.1 ²	1.9	3.0	4.9	3.7	2.8	4.0	5.5
T6.1	10.0 ³	11.0	10.8	10.7	11.2	11.1	11.1	9.3
T6.2	3.9 ²	6.1	5.4	5.9	5.4	5.3	2.4	6.0
T6.3	3.6 ²	4.3	4.2	4.7	5.0	4.3	4.0	5.3

Table 3: Rewards per task and agent, rounded to the nearest number with one decimal digit (¹GP-SARSA, ²Episodic Natural Actor-Critic, ³DQN).

Figure 6 shows the reward obtained by each agent after evaluating on 500 test dialogs after having trained on 1000, 2000, 3000 and 4000 dialogs. Each evaluation result was obtained by averaging the agent’s rewards across all tasks and the five different random seeds.

Some test dialogs between a real human and several trained policies were recorded to evaluate the quality of the learned policies and to give some context to the corresponding numbers (see tables 6, 7 and 8). Since the more easy tasks like for example T1.1 without noise and with action masking mechanism enabled reach comparable scores to the combined agent and do not model real-world scenarios adequately, task T4.3 was chosen. It contains some noise and does not rely on the action masking mechanism. Also, it assumes a cooperative user. In this interactive scenario, the user tried to keep their responses identical for all tested policies as long as possible.

Task	best PyDial	DQN	combined	-distributional	-double	-prioritization	-dueling	handcrafted
T1.1	99.4 ¹	99.2	98.2	98.8	98.8	90.2	94.4	100
T1.2	96.1 ¹	83.9	89.0	95.0	78.0	57.7	80.0	98.2
T1.3	91.4 ²	79.6	93.7	95.1	91.7	82.9	91.5	97.0
T2.1	96.8 ¹	90.8	97.7	98.1	97.6	98.2	94.7	100
T2.2	91.9 ¹	90.7	94.4	92.0	92.2	92.7	93.7	98.2
T2.3	82.3 ¹	64.3	91.2	90.1	88.8	85.2	91.3	97.0
T3.1	95.1 ¹	96.3	97.7	97.2	98.2	98.4	97.8	96.7
T3.2	84.6 ²	85.1	88.6	91.9	88.4	85.4	70.2	90.0
T3.3	76.6 ²	77.6	84.6	87.7	79.8	79.5	79.6	89.6
T4.1	91.5 ¹	83.6	93.0	93.0	95.4	93.6	93.2	96.7
T4.2	81.6 ¹	85.5	89.7	87.0	85.4	83.1	71.3	90.9
T4.3	72.1 ¹	54.6	88.0	92.2	84.5	86.1	78.3	89.6
T5.1	93.8 ¹	92.0	97.5	95.9	97.5	97.8	97.2	95.9
T5.2	74.7 ¹	75.9	82.2	87.6	85.9	82.4	78.4	87.7
T5.3	75.8 ²	59.7	66.2	77.9	71.8	62.7	71.8	85.1
T6.1	89.6 ¹	92.9	92.6	91.8	93.8	93.1	93.9	89.6
T6.2	66.7 ²	77.9	73.7	77.8	74.6	74.6	59.5	79.0
T6.3	64.6 ²	69.4	68.2	72.3	72.3	67.6	65.3	76.1

Table 4: Success rates per task and agent, rounded to the nearest number with one decimal digit (¹GP-SARSA, ²Episodic Natural Actor-Critic, ³DQN).

Written language input is handled by PyDial’s regular expression semantic input parser, responses are created by PyDial’s rule-based semantic output generator.

The combined agent was additionally trained and evaluated with multi-step targets (figure 12) and noisy linear layers for exploration (figure 13) on the OpenAI Gym cart-pole environment to prove that the implementation is working. All results for this environment use a maximum turn count of 200 and were obtained by averaging across 15 testing episodes every 10th training episode with the hyperparameters from table 13.

6.4 Discussion

Best overall reward and accuracy was obtained by the combined agent without distributional learning method. This may seem somewhat surprising

Agent	CR		SFR		LAP		All	
	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.	Suc.	Rew.
best PyDial	94.4% ¹	11.3 ³	81.7% ¹	6.9 ¹	69.4% ²	4.3 ²	80.7%	7.3
DQN	92.5%	11.3	83.2%	8.1	67.5%	4.7	84.1%	8.0
combined	96.1%	12.3	86.3%	9.0	82.0%	7.8	88.1%	9.7
- distributional	95.8%	12.2	88.5%	9.4	86.1%	8.2	90.1%	9.9
- double	96.9%	12.4	84.1%	8.4	81.5%	7.7	87.5%	9.5
- prioritization	95.2%	12.2	79.3%	7.7	77.4%	7.1	84.0%	9.0
- dueling	95.2%	11.9	75.5%	6.6	79.7%	7.1	83.5%	8.5
handcrafted	96.5%	11.5	90.8%	9.2	89.1%	8.6	92.1%	9.8

Table 5: Success rates and rewards per agent, averaged over all environments and rounded to the nearest number with one decimal digit (¹GP-SARSA, ²Episodic Natural Actor-Critic, ³DQN).

at first, since other benchmarks such as Atari 2600 reported distributional agents outperforming non-distributional ones (Hessel et al., 2017). A possible explanation could be that the amount of training dialogs was simply too small to show a benefit from the distributional reinforcement learning method. Overall difference in reward to the distributional agent is only 0.2 and 2% in success rate. This finding coincides with the reports of the Rainbow agent, where Hessel et al. (2017) found that distributional learning performed similar to non-distributional learning for the first 40 million frames on the Atari 2600 benchmark, only after crossing this mark did performance surpass the non-distributional method.

The biggest performance increase is gained by including the dueling network architecture. According to theory, the implication is that in the domain of dialog systems there should be a number of states where action choice does not matter much. Such states should calculate a simple advantage function, where simple is meant in the sense that it should not compute a function with significantly different values for each action but rather a function with few different values, or at least very similar values in the approximation setting here. The existence of such states seems justified, for example in the

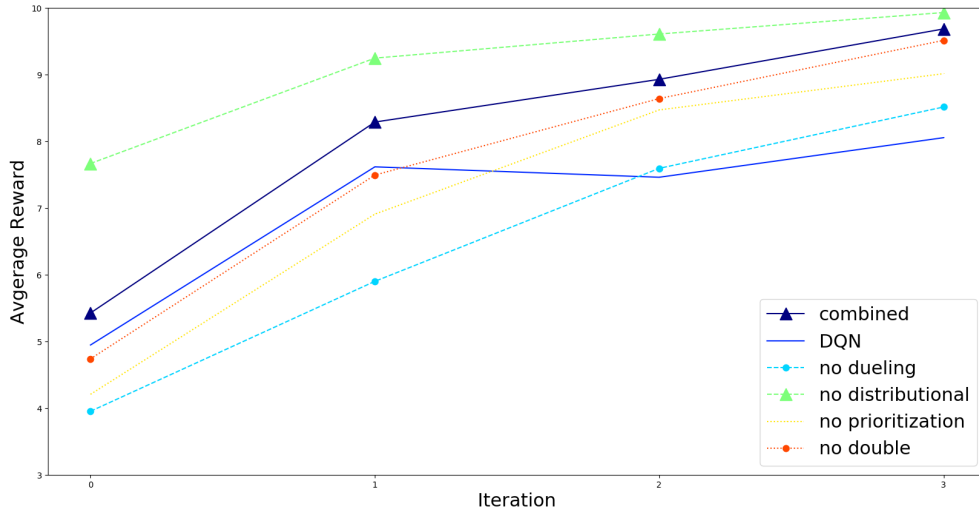


Figure 6: Average reward for policies evaluated on 500 dialogs after training with 1000 (iteration 0), 2000 (iteration 1), 3000 (iteration 2) and 4000 (iteration 3) dialogs.

beginning where a user has just greeted the system but not yet given any search criteria, the only actions that seem reasonable are information gathering requests or confirmations in case the system already misunderstood the first input. One of the worst actions is expected to be the bye-action because it would immediately result in a terminated dialog with reward -1 and no further chance of improvement. Regarding the Cambridge Restaurants ontology, it should not be overly important which information is requested first by the system since most of the time it should require to fill all possible slots in order to select a fitting restaurant. Or near the end of a dialog where the system already gathered all necessary information, the system should only have to choose between informing the user, if any result matches the user's restrictions and requesting alternatives if not. There could be many more states like that, but we focus on the scenario of the dialog start without user provided information because it is easy to reproduce with all policies since dialog has not branched yet.

Turn	Actor	Utterance
1	System	Hello, welcome to the laptop information system, what kind of laptop are you after?
1	User	Hi, I'm looking for a cheap and light laptop
2	System	Would you like the hard drive to be small, medium, or large?
2	User	Medium should suffice
3	System	Sorry would you like a product for business computing or not?
3	User	Yes, please
4	System	The satellite-pro-nb10-a-10p is in the budget price range, is in the light weight range and is in the medium drive range.
4	User	Thank you, bye
5	System	Thank you, goodbye.

Table 6: Recorded dialog between a human and the trained combined agent for task T4.3.

In the following, both the combined agent and the non-dueling agents trained on the Cambridge Restaurants ontology without actions masks and without noisy inputs were given only a “Hi” as user input. Action masks were turned off as not to influence the resulting value distribution. The corresponding Q-values were grouped using Affinity Clustering (Frey and Dueck, 2007) with preference -0.001 .

Figure 7 shows three detected clusters for the combined agent, where group 1 contains all request actions, which is exactly what was to be expected. Group 2 contains all select and confirm actions except the confirmation for food type. Group 3 consists of the confirmation of food type along with inform actions, repetition prompt and dialog ending actions such as saying goodbye or restarting the dialog.

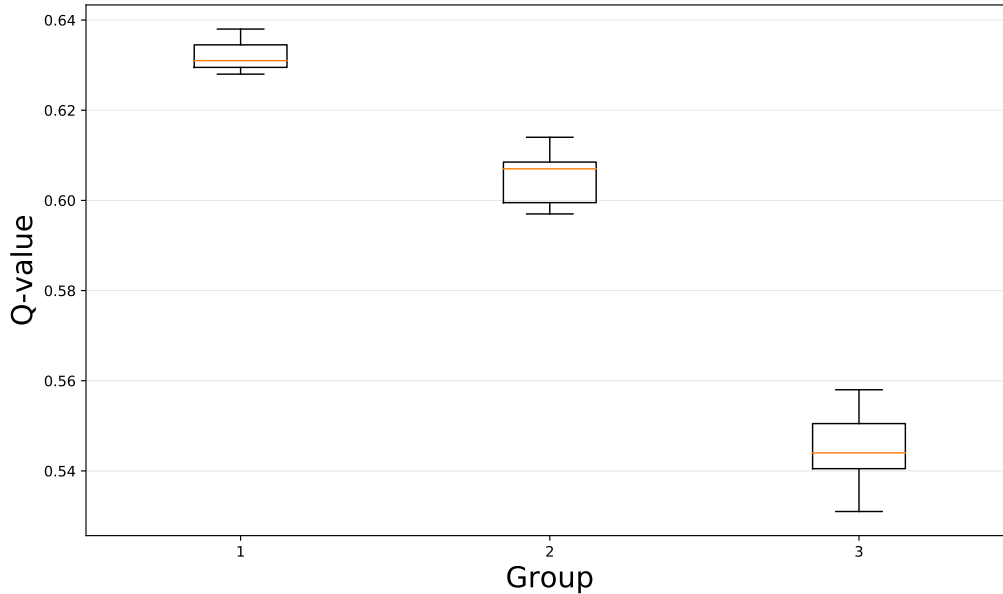


Figure 7: Clustered Q-values from the combined agent for task T2.1 after the first turn where the user said "Hi". Values are rounded to 3 decimal digits.

Q-value clustering for the non-distributional agent is displayed in figure 8, where the same clustering process yielded 6 clusters. Information gathering actions are distributed across the first three clusters with the first cluster containing food type, the second price range and the third area requests. Clusters here are less homogenous compared to the combined agent, meaning they contain very different types of actions, and cluster centers lie further apart, confirming at least for this example the theory that the advantage function might result in simpler functions for states where many actions may produce similar outcomes.

The situation was found to be similar for the case where the system is already in possession of sufficient information to inform the user about a matching result. These results could explain part of why dueling agents perform much better for dialog policy learning than the non-dueling agent.

Ranking second in terms of reward improvement is the prioritized ex-

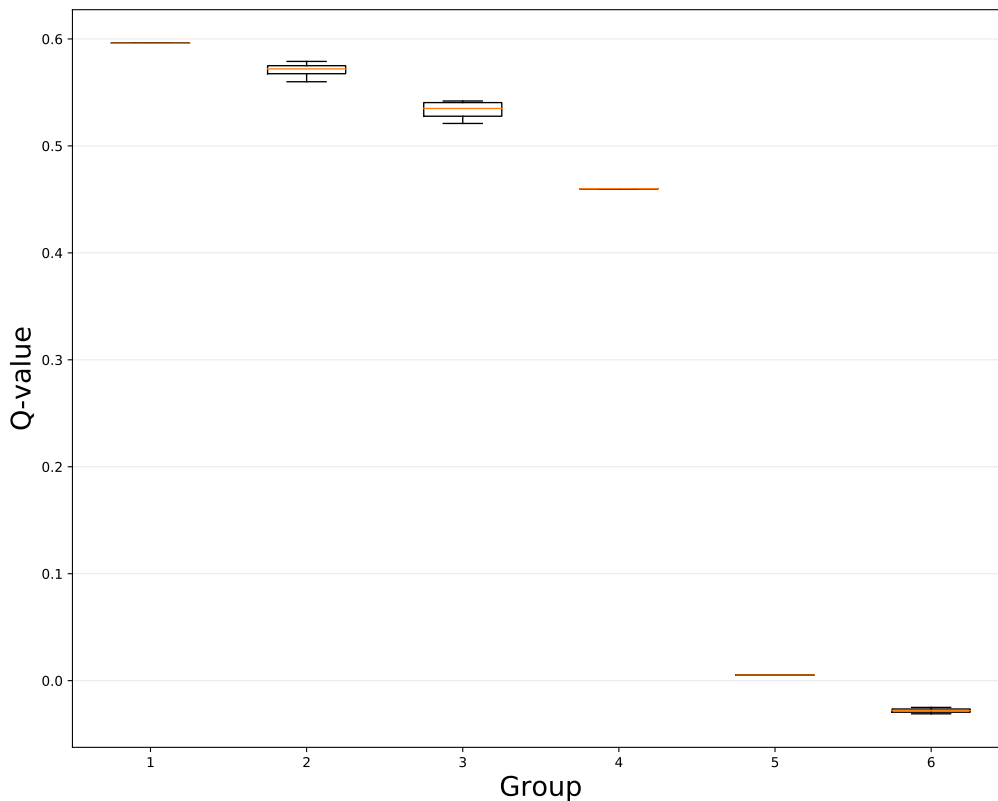


Figure 8: Clustered Q-values from the non-distributional agent for task T2.1 after the first turn where the user said "Hi". Values are rounded to 3 decimal digits.

perience replay. Table 5 shows that the performance drop when omitting prioritization is not as severe as in the non-dueling case, but still quite high. The effect of prioritization is probably more noticeable because of the rather small amount of training episodes. It remains to be seen if the resulting rewards from training for a longer period would show a similar difference.

Including a periodically updated target net and using it for action evaluation as in the double DQN method had the least impact on the agent's final rewards, nevertheless it represents an improvement on overall performance.

The update frequency used here is quite high with the target network being updated every six dialogs, but lower frequencies seemed to result in worse agent performance. Even though the agent without a target network outperforms the combined agent in some tasks (see tables 3 and 4), overall performance is more stable in the combined agent and the non-distributional agent when averaged across tasks (compare with table 5), so the maximization bias does seem to hurt agent performance in dialog policy learning.

That the DQN implementation here performs better than PyDial’s DQN agent on average (compare with table 4 in Casanueva et al. (2017)) could be caused by the parameter choice, implementation details or both. But it underlines the importance of creating a custom implementation sharing a big part of the code with the combined agents to obtain a fair baseline.

Multi-step targets were removed from the combined agents because they hurt final agent performance in almost every task, as can be seen in table 11 and 12. These methods only managed to score comparable values to the other agents in task T1.1 and even there only with the smallest multi-step setting of 2 steps. Figure 9 shows two commonly encountered evaluation results: The rewards of the multi-step agent increase substantially between the first two iterations and afterwards they start to diminish, as shown for task T3.1, in contrast to the combined agent which keeps improving. Or, as the second scenario, the multi-step agent seems not to learn at all, exemplary shown for task T4.1. In general, the bigger the steps were chosen, the worse the obtained rewards.

To minimize the possibility that multi-step learning performed bad simply because of a faulty implementation, the same agent code was used for tests on the cart-pole environment by OpenAI Gym. There, the combined agent clearly benefits from multi-step targets and reaches maximum performance quicker with increasing multi-step size as can be seen in figure 12. So either multi-step targets do not work well in the domain of dialog systems since it worked fine in the OpenAI Gym testing environment or the tested multi-step sizes of 2,3 and 5 did not work in combination with the chosen parameters.

Especially reward curves like for task T4.1 (see figure 9) seem to indicate the latter. Some parameter like the learning rate could be inadequate since neither improvement nor worsening was achieved and the agent seemed not to change at all. On few tasks like for example T1.1 and T1.2, performance comparable to the other agents was achieved, at least for smaller step sizes. This is emphasized by comparing the training loss between a two-step and five-step agent as demonstrated in figure 10 and 11 for task T1.1, which increases with increasing step size and correspondingly increases the gradients, possibly leading to unstable learning.

The same holds for exploration by noisy linear layers. Prove for a working implementation is given in figure 13, where the agent manages to successfully reach maximum performance. But, comparing the amount of episodes required to reach final performance, the noisy exploration agent seems to be much slower than the multi-step methods in the same environment (figure 12). Unfortunately, capping the maximum turn count prohibits any insightful information into the possible height of the rewards the noisy agent could have reached in comparison to other agents after reaching the turn cap. Dialog policy learning rewards and success rates for noisy exploration can be found in tables 9 and 10. Again, increasing the amount of training episodes could have been helpful. In the current setting, it seems that especially the combination of noisy environments with noisy exploration does not work too well (compare for example Tasks T1.1 and T3.1 which differ only in the amount of environmental noise). The initialization of the noise also seems to have a big influence on final performance, with 0.5 seeming too high and 0.1 too low.

Table 5 shows that each combined agent improves upon the basic DQN implementation. Regarding learning speed with respect to evaluation reward, figure 6 highlights the combined and the non-distributional agents as the fastest learning methods, surpassing the final rewards of the DQN agent and the agents without prioritized replay and dueling network architecture after training on 2000 dialogs already. While the combined agent and the

agent without double Q-learning almost reach the performance of the non-distributional agent, they do so only in the end after having trained on the full set of 4000 dialogs. Reward-wise, the combined agent outperforms the best PyDial agent in 16 out of 18 tasks. When averaged across all tasks, the combined agent and the combined agent without distributional learning performed best with the first showing a reward increase of more than 32% and the second an improvement of more than 35% over the best PyDial agent (GP-SARSA, mean reward 7.3) and more than 20% and 23% over the vanilla DQN. Both the combined agent and the non-distributional agent prove to be on par with handcrafted policies.

Quality-wise, sample dialogs in tables 6, 7 and 8 show that the combined agent produces dialog strategies similar to the handcrafted policy. They have the same amount of turns and only differ in one question. While the combined agent ensures that the user really chose a cheap product, the handcrafted policy asks an additional information gathering question, namely for battery rating. But this question was not necessary for the combined agent, which nevertheless reported the same result to the user. This example could indicate that indeed more efficient dialog policies could be learned by reinforcement learning methods. Maybe longer training would have removed the question if the user really wanted a product from the budget price range, which would result in one turn less for this example dialog compared to the dialog obtained by talking to the handcrafted policy. The best PyDial agent (DQN) for this example dialog did not even complete the dialog successful, since it seemed to be caught in an infinite loop and the user decided to end the dialog after eight turns.

7 Conclusion and Future Work

In this thesis, a basic DQN agent was implemented and augmented with various extensions following the procedure from the Rainbow agent (Hessel et al., 2017), resulting in a combined agent. In contrast to Rainbow, the use

of the categorical distributional learning algorithm (Bellemare et al., 2017) was replaced by the newer and more theory-consistent quantile regression algorithm (Dabney et al., 2017).

Following a parameter search and some initial evaluations, the agent was further modified by swapping the noisy linear layers for exploration (Fortunato et al., 2017) with a simple ϵ -greedy scheme and the multi-step targets (Sutton, 1988) with standard one-step targets. Both extensions were found not to work well in this dialog system environment. Nevertheless, the implementation seemed to work when applied to the OpenAI Gym cart-pole testing environment Brockman et al. (2016). Future experiments might consider longer training periods to assess whether a large enough reward increase could be achieved to justify the additional time requirement. The same could be tested for the agents relying on distributional reinforcement learning.

The combined agent was trained and evaluated on different dialog ontologies using different noise levels and user behaviors to investigate its generalization capabilities and to test it in more realistic, real-world related scenarios. Results show that all of the tested agents improve on the best PyDial agents (Casanueva et al., 2017) with respect to averaged rewards over all tasks. The combined agent without distributional learning method was found to perform best, but only marginally better than the combined agent. Rewards of both agents are on par with the handcrafted policies from PyDial.

Dosovitskiy and Koltun (2016) recently found promising deep reinforcement learning results by choosing actions according to a linear combination of multiple value predictions of different lookahead horizon sizes. This could be an interesting addition for a further extended combined agent.

Another recent development, the implicit quantile regression algorithm (Dabney et al., 2018) could also be tested instead of the quantile regression algorithm used here. It randomly chooses probabilities as additional network input instead of calculating outputs over a fixed set of probabilities.

Lastly, the integration of hierarchical reinforcement learning could prove

beneficial. Casanueva et al. (2018) obtained very good results using hierarchical methods. Since one of their methods, called FDQN, is based on DQN, it would be interesting to evaluate possible improvements by embedding the combined agent presented here in a hierarchical scheme.

Turn	Actor	Utterance
1	System	Hello, welcome to the laptop information system, what kind of laptop are you after?
1	User	Hi, I'm looking for a cheap and light laptop
2	System	Would you like the hard drive to be small, medium, or large?
2	User	Medium should suffice
3	System	Sorry would you like a product for business computing or not?
3	User	Yes, please
4	System	Did you say you are looking for a laptop in the budget price range?
4	User	Yes
5	System	Did you say you are looking for a laptop in the budget price range?
5	User	Yes, cheap
6	System	Did you say you are looking for a laptop in the budget price range?
6	User	Budget
7	System	Sorry would you like something in the small drive size range or in the medium drive size range?
7	User	Medium
8	System	Did you say you are looking for a laptop in the budget price range?
8	User	bye
9	System	Thank you, goodbye.

Table 7: Recorded dialog between a human and the best PyDial agent (DQN) for task T4.3.

Turn	Actor	Utterance
1	System	Hello, welcome to the laptop information system, what kind of laptop are you after?
1	User	Hi, I'm looking for a cheap and light laptop
2	System	Would you like your battery rating to be standard, good, or exceptional?
2	User	I don't care
3	System	Would you like the hard drive to be small, medium, or large?
3	User	Medium should suffice
4	System	The satellite-pro-nb10-a-10p is in the light weight range, has a standard battery rating , is in the budget price range and is in the medium drive range .
4	User	Thank you, bye
5	System	Thank you, goodbye.

Table 8: Recorded dialog between a human and PyDial’s handcrafted policy for task T4.3.

Appendices

Task	noisy $\sigma_0 = 0.05$	noisy $\sigma_0 = 0.1$	noisy $\sigma_0 = 0.3$	noisy $\sigma_0 = 0.5$
T1.1	13.2	12.4	12.3	12.5
T1.2	7.5	9.1	7.7	5.0
T1.3	8.7	10.4	7.8	6.0
T2.1	6.2	4.6	10.5	8.4
T2.2	6.0	3.5	10.6	8.5
T2.3	5.5	3.8	10.2	6.0
T3.1	9.0	10.4	9.9	9.9
T3.2	5.6	7.6	6.5	3.7
T3.3	7.1	8.1	6.4	3.4
T4.1	8.9	4.6	8.1	5.3
T4.2	1.5	3.0	5.2	6.4
T4.3	3.3	3.5	6.3	6.0
T5.1	5.1	12.0	6.9	6.4
T5.2	3.8	3.6	2.0	-0.4
T5.3	3.7	1.5	0.8	-0.4
T6.1	8.9	9.3	7.7	7.1
T6.2	4.4	5.1	3.3	2.1
T6.3	4.8	5.1	3.8	2.2
Avg.	6.3	6.5	7.0	5.4

Table 9: Rewards for the combined agent with noisy linear layers for exploration, initialized with different noise levels σ_0 and rounded to the nearest number with one decimal digit.

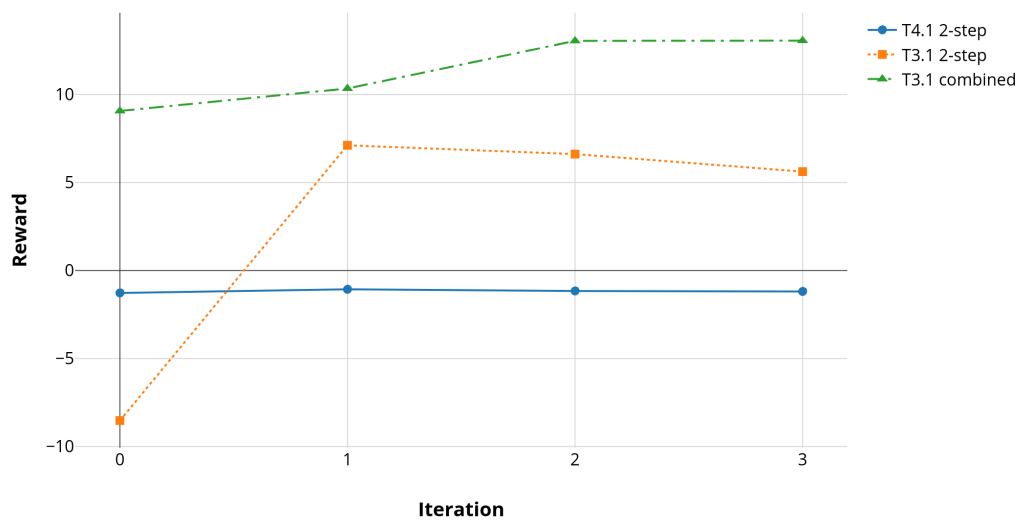


Figure 9: Rewards for the multi-step agent with step size 2 and the combined agent for different tasks.

Task	noisy $\sigma_0 = 0.05$	noisy $\sigma_0 = 0.1$	noisy $\sigma_0 = 0.3$	noisy $\sigma_0 = 0.5$
T1.1	96.6	92.6	92.3	93.4
T1.2	71.7	80.4	74.2	59.4
T1.3	78.1	88.4	72.7	63.3
T2.1	55.6	45.3	92.0	84.4
T2.2	67.5	41.8	91.5	85.9
T2.3	68.6	50.4	90.3	76.0
T3.1	80.3	85.9	83.2	83.6
T3.2	68.3	78.2	72.1	57.0
T3.3	77.4	82.0	71.2	51.2
T4.1	86.4	47.4	72.1	63.4
T4.2	39.7	51.6	61.1	78.0
T4.3	59.3	52.2	65.1	73.5
T5.1	68.1	98.0	75.2	72.6
T5.2	68.0	65.7	56.1	40.9
T5.3	68.9	53.3	48.8	39.9
T6.1	83.2	85.6	76.2	73.2
T6.2	67.4	72.2	60.7	53.1
T6.3	71.2	71.0	64.6	52.4
Avg.	70.9	69.0	73.3	66.7

Table 10: Success rates for the combined agent with noisy linear layers for exploration, initialized with different noise levels σ_0 and rounded to the nearest number with one decimal digit.

Task	2-step	3-step	5-step
T1.1	13.4	12.0	5.8
T1.2	10.4	6.7	-2.2
T1.3	7.2	3.9	-2.3
T2.1	-1.1	-1.6	-1.3
T2.2	-2.5	-3.7	-2.2
T2.3	-2.4	-2.9	-4.0
T3.1	9.4	3.3	1.7
T3.2	0.4	-3.6	-7.1
T3.3	-1.2	-3.7	-6.6
T4.1	-1.0	-1.1	-1.7
T4.2	-1.5	-1.8	-1.3
T4.3	-3.8	-2.6	-4.6
T5.1	-1.6	-0.2	-3.4
T5.2	-5.7	-6.7	-6.9
T5.3	-6.4	-6.3	-6.6
T6.1	1.8	1.5	-0.5
T6.2	-3.2	-6.7	-7.4
T6.3	-4.4	-6.0	6.9
Avg.	0.4	-1.1	-3.2

Table 11: Rewards for the combined agent with multi-step targets, rounded to the nearest number with one decimal digit.

Task	2-step	3-step	5-step
T1.1	96.5	90.3	61.5
T1.2	88.0	68.9	21.1
T1.3	71.8	54.5	22.2
T2.1	1.0	0.9	2.5
T2.2	0.4	7.1	7.7
T2.3	6.7	5.1	10.7
T3.1	79.5	50.5	42.1
T3.2	38.0	16.2	0.0
T3.3	28.7	15.7	0.0
T4.1	0.7	0.1	0.1
T4.2	4.7	0.1	7.4
T4.3	0.3	3.7	1.1
T5.1	27.1	33.6	16.5
T5.2	5.6	1.5	0.0
T5.3	4.5	1.4	0.0
T6.1	43.7	43.2	31.1
T6.2	22.8	0.9	0.0
T6.3	12.6	4.7	0.0
Avg.	20.8	14.2	5.7

Table 12: Success rates for the combined agent with multi-step targets, rounded to the nearest number with one decimal digit.

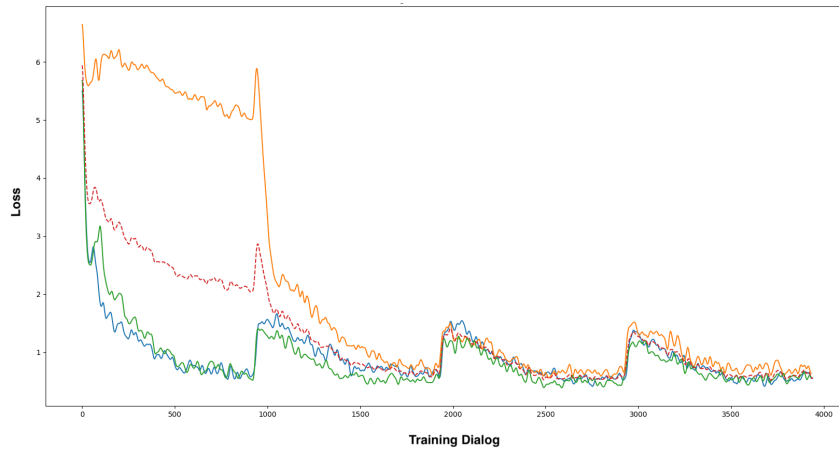


Figure 10: Train loss of the multi-step agent for task T1.1 with step size 2.

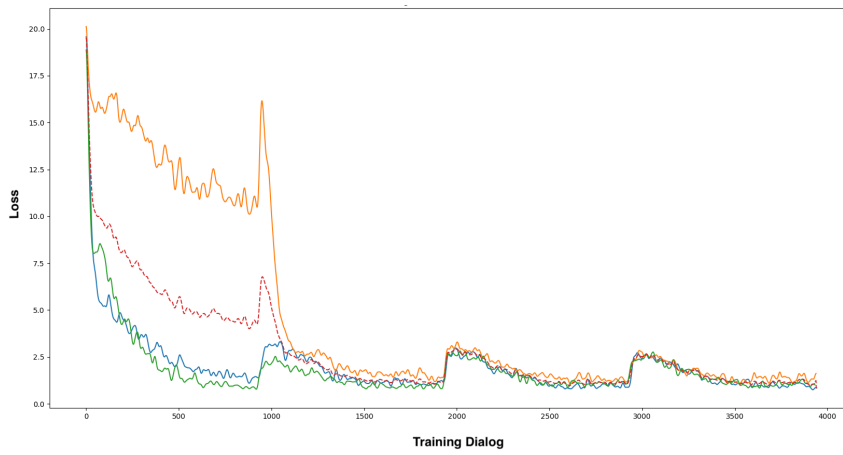


Figure 11: Train loss of the multi-step agent for task T1.1 with step size 5.

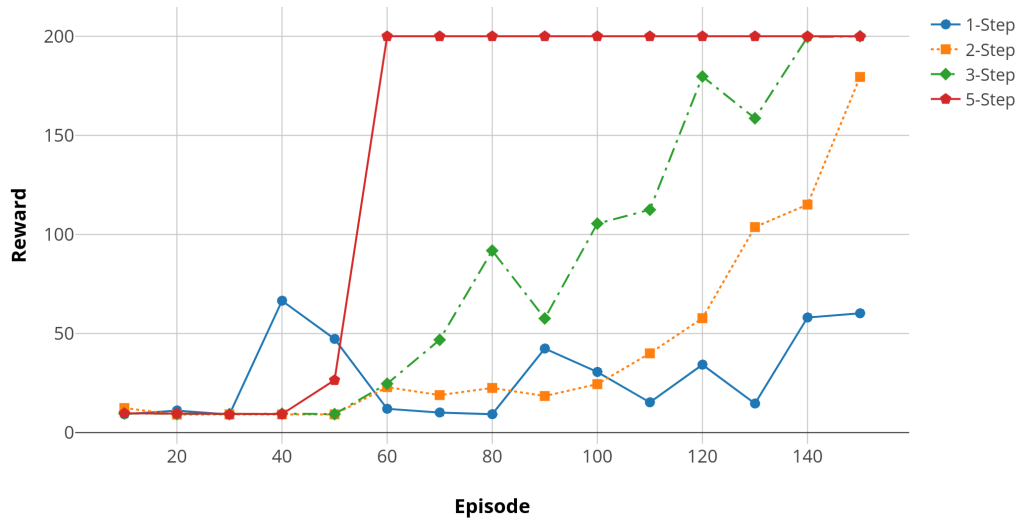


Figure 12: Results for the combined agent on the OpenAI Gym cart-pole environment for different multi-step sizes. Maximum turns were capped at 200.

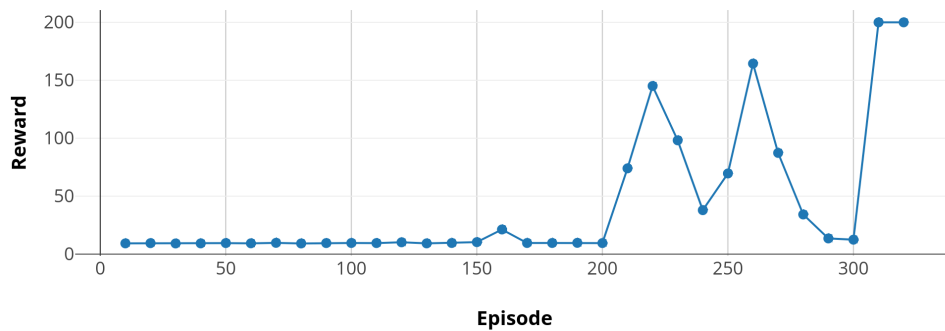


Figure 13: Results for the combined agent on the OpenAI Gym cart-pole environment for noisy linear exploration with initial noise parameter $\sigma_0 = 0.1$. Maximum turns were capped at 200.

parameter name	parameter value
learning rate	0.01
gradient clipping	5.99 (L2-Norm)
exploration ϵ -decay	linear from 1.0 \rightarrow 0.0 (figure 12)
initial layer noise σ_0	0.1 (figure 13)
replay buffer size	4096 (prioritized)
prioritization exponent α	0.5
importance sampling weight β	linear from 0.6 \rightarrow 1.0
multi-steps	{1,2,3,5} (figure 12) / 5 (figure 13)
discount factor γ	0.99
batch size	16
atoms	25
target net copy frequency	5
architecture	dueling
shared layer neurons	16
value layer neurons	32, 32
advantage layer neurons	32, 32

Table 13: Hyperparameters used for producing figures 12 and 13.

Bibliography

- Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.
- Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.
- Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.
- Daniel G Bobrow, Ronald M Kaplan, Martin Kay, Donald A Norman, Henry Thompson, and Terry Winograd. Gus, a frame-driven dialog system. *Artificial intelligence*, 8(2):155–173, 1977.
- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
- Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. A benchmarking environment for reinforcement learning based task oriented dialogue management. *arXiv preprint arXiv:1711.11023*, 2017.
- Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Stefan Ultes, Lina Rojas-Barahona, Bo-Hsiang Tseng, and Milica Gašić. Feudal reinforcement learning for dialogue management in large domains. *arXiv preprint arXiv:1803.03232*, 2018.
- Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. *arXiv preprint arXiv:1710.10044*, 2017.
- Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*, 2018.

- Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. Towards end-to-end reinforcement learning of dialogue agents for information access. *arXiv preprint arXiv:1609.00777*, 2016.
- Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
- Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.
- Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.
- Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.
- Milica Gasic and Steve Young. Gaussian processes for pomdp-based dialogue manager optimization. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 22(1):28–40, 2014.
- Milica Gašić, Filip Jurčiček, Simon Keizer, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. Gaussian processes for fast policy optimisation of pomdp-based dialogue managers. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 201–204. Association for Computational Linguistics, 2010.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning (adaptive computation and machine learning series). *Adaptive Computation and Machine Learning series*, page 800, 2016.
- Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

- Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.
- Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh PA, 1993.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Yun-Nung Chen, and Kam-Fai Wong. Adversarial advantage actor-critic model for task-completion dialogue policy learning. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6149–6153. IEEE, 2018.
- Martin Riedmiller. Neural fitted q iteration—first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.
- Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152. Association for Computational Linguistics, 2007.
- Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
- James E Smith and Robert L Winkler. The optimizer’s curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52(3): 311–322, 2006.
- Pei-Hao Su, David Vandyke, Milica Gasic, Dongho Kim, Nikola Mrksic, Tsung-Hsien Wen, and Steve Young. Learning from real users: Rating dialogue success with neural networks for reinforcement learning in spoken dialogue systems. *arXiv preprint arXiv:1508.03386*, 2015.
- Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press Cambridge, 1998.
- Stefan Ultes, Lina M. Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gasic, and Steve Young. PyDial: A Multi-domain Statistical Dialogue System Toolkit. In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78, Vancouver, Canada, July 2017. Association for Computational Linguistics.

- Eric Van den Steen. Rational overoptimism (and other biases). *American Economic Review*, 94(4):1141–1151, 2004.
- Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.
- Marilyn A Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12:387–416, 2000.
- Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.
- Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King’s College, Cambridge, 1989.
- Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.
- Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016a.
- Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016b.
- Jason Williams, Antoine Raux, and Matthew Henderson. The dialog state tracking challenge series: A review. *Dialogue & Discourse*, 7(3):4–33, 2016.
- Wayne Xiong, Lingfeng Wu, Fil Allea, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. The microsoft 2017 conversational speech recognition

system. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5934–5938. IEEE, 2018.

Erklärung (Statement of Authorship)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und dabei keine andere als die angegebene Literatur verwendet habe. Alle Zitate und sinngemäßen Entlehnungen sind als solche unter genauer Angabe der Quelle gekennzeichnet. Die eingereichte Arbeit ist weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen. Sie ist weder vollständig noch in Teilen bereits veröffentlicht. Die beigefügte elektronische Version stimmt mit dem Druckexemplar überein.¹

(Dirk Väth)

¹Non-binding translation for convenience: This text is the result of my own work, and any material from published or unpublished work of others which is used either verbatim or indirectly in the text is credited to the author including details about the exact source in the text. This work has not been part of any other previous examination, neither completely nor in parts. It has neither completely nor partially been published before. The submitted electronic version is identical to this print version.