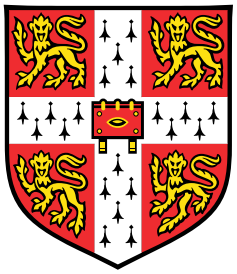
Curiosity-Driven Reinforcement Learning for Dialogue Management



Paula Wesselmann

Department of Engineering
University of Cambridge

MPhil in Machine Learning,
Speech and Language Technology

Jesus College

August 2018

Declaration

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

Word count: 13900

Paula Wesselmann
August 2018

Acknowledgements

I would like to thank my supervisors Milica Gašić and Yen-Chen Wu for help and guidance. Also I want to thank the rest of the Dialogue Systems Group office for helping me out and accepting me into the office group. And thank you to Rowing and 24MC, who made it possible for me to study at Cambridge and brought me through the year!

Abstract

Obtaining an effective reward signal for dialogue management is a non trivial problem. Real user feedback is inconsistent and often even absent. This thesis investigates the use of intrinsic rewards for a reinforcement learning based dialogue manager in order to improve policy learning in an environment with sparse rewards and to move away from inefficient random ε-greedy exploration. In addition to rewards given by a user simulator for successful dialogues, intrinsic curiosity rewards are given in the form of belief-state prediction errors generated by an intrinsic curiosity module within the dialogue manager. We investigate two main settings for this method: (1) predicting the raw next belief-state, and (2) predicting belief-states in a learned feature space. In order to meet the right difficulty level for the system to be able to learn a feature space, the model is pre-trained on a small pool of dialogue transitions. For both settings, results comparable to and better than simple ε-greedy exploration are achieved. (1) is able to learn faster, but (2) achieves higher final results and has more potential for improvements and to be successful in larger state-action spaces, where feature encodings and generalization are beneficial.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

ACER	Actor-Critic with Experience Replay
AI	Artificial Intelligence
ASR	Automatic Speech Recognition
CI	Confidence Interval
CNN	Convolutional Neural Network
CR	Cambridge Restaurant
DM	Dialogue Management
DNN	Deep Neural Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ELU	Exponential Linear Unit
GP	Gaussian Process
GRU	Gated Recurrent Unit
ICM	Intrinsic Curiosity Module
IS	Importance Sampling
KL	Kullback-Leibler Divergence
LSTM	Long Short Term Memory

MDP Markov Decision Process

MLP Multilayer Perceptron

MSVE Mean Squared Value Error

NLU Natural Language Understanding

NN Neural Network

POMDP Partially Observable Markov Decision Process

RL Reinforcement Learning

RNN Recurrent Neural Network

RV Random Variable

SDS Spoken Dialogue System

SER Semantic Error Rate

TD Temporal Difference

TRPO Trust Region Policy Optimization

Chapter 1

Introduction

Advances in artificial intelligence (AI) allow for technologies that display more human-like behavior, such as speaking robots. Initially dialogue systems were using inflexible handcrafted decision rules, which brings limitations and does not allow for these systems to behave intelligently. Reinforcement learning (RL) allows learning from interaction by maximizing rewards [35], thus removing the need for manual rules. The problem is that these rewards are often noisy, sparse or completely non-existent. To alleviate this problem, the use of *intrinsic rewards*, generated by the system itself, is suggested [15], [11].

An agent learning via RL, learns to behave in order to maximize the expected sum of rewards. The behavior is learned in form of a policy that aims to map belief-states to optimal actions. Usually rewards are given to the system externally, reinforcing good behavior while punishing bad behavior. This can be illustrated by a learning child for example, which gets rewarded by its parents: For good table manners the child gets to eat dessert, but if the child is misbehaving during dinner, it is the children's turn to do the washing up. The child learns to behave such that the positive reward, of getting dessert, is maximized.
In our case the child is a spoken dialogue system, learning to assist the user to reach a specific goal. The external rewards for learning have to come from the users of the dialogue system, giving feedback about the quality of the dialogue. Users often do not like to give feedback and have different perceptions of good and bad, which results in the rewards received by the system being sparse and inconsistent. In order to still learn desirable behaviors the system needs another reward source. Back to the analogy of the child, the child now is playing on its own, receiving no feedback by its parents that could be guiding the child's actions. The child's behavior now is driven by curiosity, an intrinsic motivation, leading the child to try out new things and explore its environment.

This thesis aims to apply estimation of intrinsic rewards in a statistical spoken dialogue system (SDS). Intrinsic rewards are beneficial to such systems, because they help to overcome the sparse and inconsistent user feedback problem and should thereby help them to move from the exclusive use of simulated dialogues to real user interactions for learning.

The approach for the work in this thesis is taken from Pathak et al. [21], who use an intrinsic curiosity module (ICM) that generates an intrinsic reward signal to drive learning. They evaluate their ICM on *VizDoom* and *Super Mario Bros*, working with high dimensional visual inputs. Pathak et al. achieve results significantly outperforming methods without intrinsic reward signals. We are using their same idea of state prediction error as an intrinsic reward signal for the dialogue management (DM) module of the *PyDial* dialogue system. If the system is able to correctly predict what belief-state it will be in after performing an action, that action was not curious and no reward is given. But if the system learns something new, that is, it was not able to predict the next belief-state, the prediction error is given as a reward for performing a curious action.

Furthermore, in RL, the environment is often explored employing an ε-greedy policy, where $\varepsilon\%$ of the time an action is chosen at random and $(1-\varepsilon)\%$ of the time the action is chosen greedily. This is done in order to balance exploration and exploitation. Using curiosity as intrinsic motivation, however, does not require random exploration since the agent now acts with the aim to explore new belief-states to maximize its reward. Efficient exploration is key for data efficient learning in large state-action spaces.

The rest of this thesis is organized as follows: Chapter 2 provides background information about SDS, the main concepts of RL, and an introduction to intrinsic motivation. Chapter 3 describes the method of state prediction error which is used for intrinsic motivation in the experiments described in Chapter 4. Finally, Chapter 5 concludes and summarizes the work as well as discusses future work on the topic.

Chapter 2

Background

2.1 Statistical Spoken Dialogue Systems

A SDS is composed of different modules, as illustrated in Figure 2.1, each module covering specific roles. SDSs are very complex since they solve many problems at once and must contend with a great deal of uncertainty. A user interacts with the dialogue system through speech and receives a spoken response.

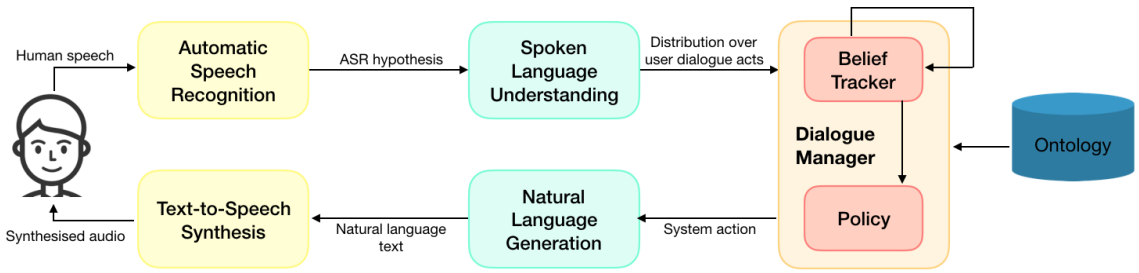


Figure 2.1 Modular architecture of a Statistical Spoken Dialogue System.

The flow of the system as illustrated in Figure 2.1 can be described as following: Using automatic speech recognition, the system builds hypothesis about what the user has said. Those hypotheses are then fed into the spoken language understanding unit which outputs a distribution over user intentions. The DM updates and tracks its understanding of the users goal, the belief-state b_t of the dialogue, at every turn t. Using RL (see Section 2.2), the DM learns a decision rule, policy , to guide the dialogue system's actions a_t, i.e. responses, given the current belief-state. To supply the user with information, the dialogue manger accesses information from an ontology for the domains of the dialogue system. The dialogue manager outputs system action a_t, which is transformed into a natural language sentence in the natural

language generation module. Via text-to-speech synthesis, the natural language sentence is then played as audio for the user.

Modules and entities of the SDS, which require further introduction for the purpose of this thesis are discussed in the the following subsections.

2.1.1 Ontology

The dialogue domain is described by the ontology. Normally this is a structured representation of the database about entities which the dialogue system can talk about. In the simplest case it consists of entities, slots, which define properties of these entities and a set of values that the slots can take, e.g. restaurant Char Sue is in the central area: name="Char Sue", area="center".

2.1.2 Dialogue Actions

Dialogue actions consist of a dialogue act type and a list of slot-value pairs. An example of a dialogue act is `inform`, matched with the slot-value pair `name="Char Sue", area="center"`, the combination makes up the dialogue act `inform(name="Char Sue", area="center")`. The SDS can perform a set of hard-coded actions, which in *PyDial* exist as a *master action set* in the ontology. In order to reduce the number of dialogue actions this *master action set* is scaled into a *summary action space* that consists of more general actions, available for the requestable and informable slots, but not specifying each of their available values. Since the *master action space* can be very large, training a policy in this action space can be difficult. Using the *summary action space* to train the policy is more efficient and shows better learning success. When a policy is trained on the *summary action space*, the selected action needs to be converted to a *master action*, which is done by a set of heuristics attempting to find optimal slots given the current belief-state [33]. The heuristics necessary to map the *summary action space* to the *master action space* need to be manually constructed for each domain.

Summary actions for the SDS *PyDial* used for this thesis include:

1. `request_`⟨`requestable slot`⟩
2. `confirm_`⟨`requestable slot`⟩
3. `select_`⟨`requestable slot`⟩
4. `inform`

5. `inform_byname`
6. `inform_alternatives`
7. `bye`
8. `repeat`
9. `reqmore`

The three requestable slots for *PyDial*s CR domain are: `food`, `area`, and `price range`, making the summary space consist of 15 actions. An example of an available summary action for the CR domain is: `inform(food)`. In the master action space this action expands to include all possible values for the `food` slot, e.g. `inform(food=indian, name ="Tiffin Truck")`. A detailed description of summary and master actions considered for this thesis can be found in Appendix A. An example dialogue, displaying summary and master actions for each turn can be found in Appendix B.

Execution Mask

In order to aid the policy training, an execution mask can be applied. The execution mask ensures that only appropriate system actions can be chosen for every belief-state, meaning that the system for example cannot issue an `inform` action the user at the beginning of the dialogue, when it has not received any information about the users goal yet. Without the execution mask, the system has to learn to not select inappropriate actions, complicating its training. The execution mask is a powerful tool to aid better policy learning, but it has to be handcrafted and is inflexible to new actions.

2.1.3 Dialogue Manager

The DM is the core component responsible for the system's behavior. It is made up of two units, the **Belief Tracker** and the **Policy**, as shown in Figure 2.1. The DM can be described as the brain of the SDS, where belief tracking is responsible for memory, and policy for making decisions, sending signals on what actions to take, which then are executed by other modules of the system.

Belief Tracker

The dialogue state b_t, which is a belief-state, denotes a representation of the system's belief of what the user wants at any point in the dialogue [33]. The user's overall goal for the dialogue is called *user goal* and may change throughout the dialogue. The user also has a short term goal for every turn of the dialogue, called the *user intent*. The *user intent* might be

to `inform` the system on what the user wants to eat or to `request` information on something.

Belief tracking accumulates evidence over the sequence of dialogue turns and adjusts the belief-state according to its observations. At each turn in the dialogue, the tracker outputs a distribution for each component of the dialogue state. There are three components of the dialogue state: goals, method, and requested slots. The goals are what the system beliefs are the user's true required values for each slot in the ontology. For slots that have not been informed yet throughout the dialogue, the goal is 'None'. Method describes how the user interacts with the system. E.g. if `inform(food=thai)`, the method becomes 'by constraints' since the user is constraining the search. Requested slots corresponds to the systems confidence of the user requesting a particular slot. It is a set of slots the user requested, of which he or she has not yet been informed.

For this thesis, a focus tracker is used, accumulating evidence over all turns in the dialogue. The focus tracker uses a model of the belief-state changing throughout the dialogue which reduces the impact of errors from the speech recognition side, compared to when a simple one-best tracker is used. To deal with the uncertainty of input in the system, the dialogues are modeled as partially observable Markov decision process (POMDP) [35], which require the belief-state to track all important information within the dialogue[1]. The belief-state update at turn t with the focus tracker is:

$$b_t(s_t = s) = \big(1 - \sum_{s'} o(s')\big) b(s_{t-1} = s) + o(s) \tag{2.1}$$

where $b_t(s_t = s)$ is the probability that the belief-state at turn t is s and $o(s)$ is the observation relating to s. The focus tracker accumulates evidence and adapts the focus of attention according to the current observation.

Policy

All system behavior comes back to the policy which regulates what actions are executed given the system's current knowledge or belief-state. Policy π is a deterministic decision rule mapping a state b_t into action a_t:

$$a_t = \pi(b_t). \tag{2.2}$$

In every turn of the dialogue, the current belief-state is the input for the policy to generate a response to the user. The policy is learned via RL and is chosen to maximize the total reward;

[1] see Section 2.2.1 for more details about POMDP

that is we chose the policy with the optimal Q-function:

$$Q(b,a) = \mathbb{E}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | b_t = b, a_t = a\right] \tag{2.3}$$

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

where T is the furthest time step in the future that is considered, t is the current turn and γ is the discount factor for future rewards. The Q-function returns the expected sum of rewards given belief-state b and action a. $\pi^*(b)$ is the optimal policy, meaning it gives the best action a to take when in belief-state b. For more detail about policy learning and the Q-function, refer to Section 2.2.

2.1.4 Reward Estimation

Adequate reward signals are essential for optimizing the dialogue policy. But what are adequate reward signals for goal oriented dialogues? A standard approach for the dialogue reward function in a goal oriented dialogue system is a per-turn penalty to encourage shorter dialogues and a large reward signal at the end of the conversation for successful dialogues. This approach is relies on user feedback, which is hard to obtain and often inconsistent. To overcome this Su et al. [28] describe an on-line active learning method in which users are asked to provide feedback about the dialog's success and this feedback then is used to estimate reward signals. A downside to their method is that it relies on labeled data to pre-train an embedding function for the dialogues. Further, when reward is only given after the conversation has finished, the system has to consider whole trajectories of conversational turns that lead to the successful completions, complicating learning. For this thesis we aim to implement an intrinsic reward signal for better state-action space exploration in a real user setting in which the extrinsic rewards are sparse and inconsistent. This aims to overcome the reward sparsity and the need of random actions for exploration. The method used for evaluation is the same in testing and training.

2.1.5 User Simulator

To train the DM via RL, a large number of sample dialogues is needed. Training with real users is expensive and proves to be more difficult. Further, when training the dialogue system from scratch, during the first iterations of learning, users must deal with bad dialogues that do not fulfill their goals. Only later on, with a system that has already learned enough, can it be beneficial for both the user and the system to learn from real user interactions. Furthermore,

feedback obtained from real users is often not consistent enough for effective RL. Instead of reliance on real user interactions, a user simulator, which resembles human behavior to some degree, can be used for efficient training of a good dialogue policy, giving consistent reward feedback.

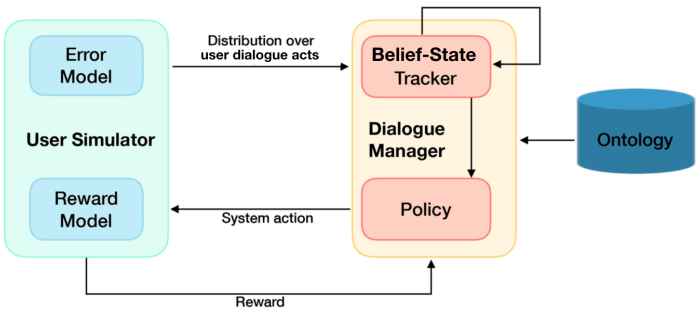


Figure 2.2 SDS with User Simulator.

The user simulator is comprised of a behavior component, the user model, to produce a semantic act, and an error simulator, to produce a list of semantic hypotheses derived from the semantic act, simulating the error coming from the Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) channel in a SDS [22]. The goal of the simulated user consists of randomly generated slot-value pairs and in some cases can change during the dialogue. The simulator's actions follow an agenda which is a dynamic stack of dialogue acts using deterministic and stochastic decisions. Additionally the simulator is responsible for providing a reward to be used for RL given the system responses and course of the dialogue. The reward is given at the end of the dialogue for successfully completed tasks only. The dynamics of a SDS using a simulated user are illustrated in Figure 2.2.

2.2 Reinforcement Learning

Reinforcement learning aims to learn how to map states to actions in order to maximize a reward signal. The agent has to balance exploring the environment and exploiting its current knowledge about the environment to do so. The interactions between agent and environment are illustrated in Figure 2.3, which closely resembles the SDS/User Simulator model of Figure 2.2. In the SDS/User Simulator model, the DM acts as agent and the user simulator represents the environment.

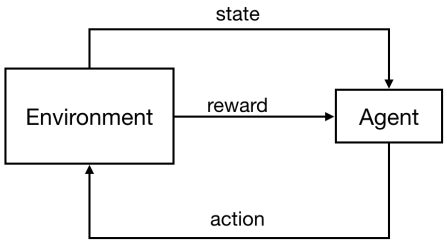


Figure 2.3 RL agent-environment interaction.

2.2.1 RL Preliminary

[2] The main elements of RL are a policy, which defines the goal of the agent (introduced in Section 2.1.3 as part of DM), a reward, a value function, defining the goal in the long run and predicting future rewards, and optionally there can be a model of the environment [29].

Markov Decision Process

In RL we assume that all states have the Markov property, which means that each state signal contains all information relevant to the system, summarizing everything important about the complete sequence of states that led to the current state [29]. Assuming Markov property for all states allows us to predict the next state given the current state and action. For a SDS state s_t is not known exactly, but a distribution over possible states called a belief-state b_t is maintained. Actions are selected based on this belief-state and the system transitions to the next state s_{t+1} based on the unobserved state s_t and action a_t. The agent observes any state s_t as observation o_t. The decision process therefore now is a partially observable Markov decision process (POMDP). Figure 2.4 shows the POMDP as a graphical model. Policy π_θ determines the system's action which will lead to the next state.

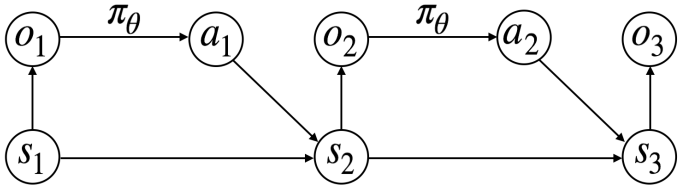


Figure 2.4 Graphical model of a partially observable Markov decision process.

Given an existing belief-state b_t, the last system action a_t, and a new observation o_{t+1}, the belief-state of the SDS is updated (see Equation 2.1).

[2] this section is based on MLSALT7 (Reinforcement Learning) lecture notes

The POMDP can be viewed as continuous space MDP in terms of policy optimization, where the states are belief-states [9]. POMDP's are good to model dialogues [34],[5],[35], since they assume noisy observations o_t instead of fully observable states, which in the case of a SDS are the uncertainties from the ASR and SLU.

Reward

The reward is feedback an agent receives after performing one or a number of actions. Rewards are given to the system when it reaches certain desired states and are used for policy learning. The goal and purpose of an agent can be thought of as the maximization of the expected value of cumulative reward over the T dialogue turns, which is called the return R.

$$R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k} \tag{2.5}$$

where $0 \leq \gamma \leq 1$ is the discount factor for future rewards. For successful learning to achieve a goal, it is important that the rewards truly indicate what is to be accomplished rather than just how to accomplish it.

Policy

As described in Section 2.1.3, the policy acts as the core element of a RL agent, determining its behavior by mapping perceived states of the environment to actions (see Equation 2.2) for the agent to take when in those states. An optimal policy maps states to actions that promise the biggest overall return (see Equations 2.3, 2.4).
The policy learning methods described in Sections 2.2.4 and 2.2.5 specify how the agent changes its policy given its experience.

Value Functions

Value functions determine how good it is for the agent to be in a particular state and are estimated in policy learning to find optimal state action pairs. Value function V is the expected value of being in a particular belief-state b:

$$V(b) = \mathbb{E}[R_t | b_t = b] \tag{2.6}$$

The state-action value function, called Q-function, is the expected value of being in a particular state and performing a particular action in that state:

$$Q(b,a) = \mathbb{E}[R_t | b_t = b, a_t = a] \tag{2.7}$$

RL and dynamic programming use the important fundamental property of value functions satisfying particular recursive relationships [29]. For any policy π and state b_t, the following consistency condition holds between $V(b_t)$ and $V(b_{t+1})$:

$$V_\pi(b) = \sum_a \pi(a,b) \sum_{b'} \sum_r p(b', r|b, a)(r + \gamma V_\pi(b')) \tag{2.8}$$

relating the value function of state b to the value function of its successor state b'. This is known as the Bellman equation. The Q-function can be expressed in form of the Bellman equation as follows:

$$Q_\pi(b_t, a_t) = \mathbb{E}_\pi \left[r_{t+1} + \gamma Q_\pi(b_{t+1}, \pi(b_{t+1})) \right] \tag{2.9}$$

In both cases $0 \leq \gamma \leq 1$ is the discount factor for future rewards.

On- and Off-Policy Learning

A behavior policy is the policy an agent is using during training to generate data. For on-policy methods, the behavior policy is the same as the learned for policy, meaning that decisions are made from the same policy as the one that is evaluated and improved.

Off-policy methods have differing behavior and learned policies. The behavioral policy generates data, while the learned policy is being optimized. Having two separate policies for behavior and learning can be beneficial, since exploration does not have to be part of the learned optimal policy used later on, the agent can take suboptimal actions while the policy is optimal.

2.2.2 Q-learning

Q-learning is a temporal difference (TD), off-policy learning method. TD methods only wait until the next time step to update the value estimates unlike waiting for the next episode. At each time $t+1$, a target is formed, and an update is made using the observed reward $r+1$ and discounted estimated for the value of being in belief-state b_{t+1}, $V(b_{t+1})$:

$$V(b_t) = V(b_t) + \alpha[r_{t+1} + \gamma V(b_{t+1}) - V(b_t)] \tag{2.10}$$

where α is the learning rate and $r_{t+1} + \gamma V(b_{t+1}) - V(b_t)$ is the TD error in the estimate at time step t. In Q-learning, the optimal action-value function, Q, is directly approximated using the TD update method:

$$Q(b_t, a_t) = Q(b_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(b_{t+1}, a) - Q(b_t, a_t)] \tag{2.11}$$

The policy getting updated is independent of the policy being followed. Pseudo code for the Q-learning algorithm is shown below:

Initialize Q arbitrarily, $Q(b_{terminal},) = 0$
repeat
    Initialize beliefstate b
    repeat
        Choose action $a = arg\max_a Q(b,a)$ (ε-greedily)
        Take action a and observe reward r and next beliefstate b'
        $Q(b,a) \leftarrow Q(b,a) + \alpha[r + \gamma \max'_a Q(b',a') - Q(b,a)]$
        $b \leftarrow b'$
    until beliefstate b is terminal
until convergence or final episode

2.2.3 Deep RL

In deep reinforcement learning (DRL) the policy and/or value function are approximated with a deep neural network (DNN) as non-linear functions. DRL allows for previously intractable decision-making problems in high dimensional state and action spaces to be solved [2]. This is because DNNs with RL enable efficient generalization over the belief space with less computational complexity by automatically finding low-dimensional representations (features) of high-dimensional data. One downside to DNNs for RL is that they do not provide uncertainty estimates, which are used for efficient exploration in RL.

2.2.4 DQN

Deep Q-networks (DQN) [16] use a DNN to approximate the optimal action-value function (Q-function): $Q^*(b,a)$, mapping a beliefstate b_t to the values of possible actions a_t [4]. DQN[3] is based on Q-learning (see Section 2.2.2). To approximate the state-action value-function Q, a NN transforms input vector $\mathbf{x}$ into an output $\mathbf{y}$, where $\mathbf{h}$ are the hidden layer NN outputs:

$$\begin{aligned} \mathbf{h}_0 &= g_0(W_0\mathbf{x}^T + b_0) \\ \mathbf{h}_i &= g_i(W_i\mathbf{h}_{i-1}^T + b_i) \\ \mathbf{y} &= g_m(W_m\mathbf{h}_{m-1}^T + b_m) \end{aligned} \tag{2.12}$$

[3] information from the MLSALT 7 lecture notes on Deep Reinforcement Learning

where $0 \leq i \leq m$, g_i is a differentiable activation function such as hyperbolic tangent or sigmoid and W_i and b_i are the parameters to be estimated. To optimize the value function, the NN is trained to minimize a loss function L. For Q-function $Q(b_t, a_t; w_t)$ represented by a DQN, where w_t is the weight vector of the NN, the loss function is the squared error between the current prediction and the one-step look-ahead value:

$$L(w_t) = \mathbb{E}\left[(y_t - Q(b_t, a_t; w_t))^2\right] \tag{2.13}$$

y_t represents the target value $r + \gamma \max_{a'} Q(b', a'; w_t)$, which can be calculated after performing action a in beliefstate b and observing reward r. The network weights w_t are updated using gradient descent:

$$w_{t+1} \leftarrow w_t + \alpha(y_t - Q(b_t, a_t; w_t))\nabla Q(b_t, a_t; w_t) \tag{2.14}$$

Using non linear function approximation such as DQN for RL is known to be unstable [12], [2] due to the correlations present in the sequence of observations, the fact that small Q updates can significantly change the policy and the correlations between Q-values and the target values $r + \gamma \max_{a'} Q(s', a')$ [16]. This instability problem is addressed by experience replay [23] and target networks. Experience replay randomizes over the data to remove correlations in the observation sequence [16]. Further experience replay increases learning speed by using batches to update NNs and avoids catastrophic forgetting by reusing past transitions [13]. The target network technique fixes parameters of the target function and only replaces them with the latest network after a number of steps. This technique is beneficial since it stabilizes the target function which otherwise is changed very frequently with DNNs.

2.2.5 ACER

Actor critic with experience replay (ACER) approximates both the policy-function π (actor) and Q-function (critic) with DNNs. The actor critic method alternates between actor improvement, which aims to improve the current policy, and critic evaluation, which evaluates the current policy with the Q-function, this is done in an off-line setting. ACER introduces experience replay for the actor critic learning method and is using DQN for Q-function approximation. DQN alone overcomes correlated states and the target problem by using Experience Replay, but it only estimates the Q-function, leading to unstable learning. The ACER combines recent advances in DRL, including experience replay [13], truncated importance sampling (IS) with bias correction [32], trust region policy optimization (TRPO) [24], and the off-policy Retrace algorithm [18]. Weisz et al. [33] show that ACER works well on

policy optimization tasks in the SDS domain, providing stable and sample efficient learning.

As mentioned in Section 2.2.4 about DQN, **experience replay** means to reuse previous interactions for learning. Experience replay is beneficial because it speeds up learning and avoids catastrophic forgetting. Experiences are sampled in random batches from a pool of past data. **Truncated importance sampling** is used to avoid vanishing or exploding trajectories. The importance sampling weights are truncated and a bias correction term is added to take errors into account. Target values are used to stabilize the target function by only replacing it after a number of steps. Targets are estimated using the **Retrace algorithm**, which is a non-biased estimator with small variance. The updated Q-function estimate Q^{ret} is computed using sampled state-action trajectories from the replay memory:

$$Q^{ret} = Q(b,a) + \mathbb{E}_{\mu}\left[\sum_{t\geq 0}\gamma^t\left(\prod_{s=1}^{t}\lambda\min(1,p(a_s|b_s))\right)(r_t+\gamma V(b_{t+1})-Q(b_t,a_t))\right] \quad (2.15)$$

where μ is the behavioral policy and λ a scaling constant. Using Q^{ret}, the loss function is modified to:

$$L(w_t) = \mathbb{E}\left[(Q^{ret}-Q(b_t,a_t;w_t))^2\right] \quad (2.16)$$

Small changes in the parameter space can lead to erratic changes in the policy, which is solved by calculating the natural gradient. Instead of calculating the computational expensive natural gradient, the gradient can be approximated as the KL divergence. **Trust-region policy optimization** is used to approximate the KL divergence between policies of subsequent parameters.

Both ACER as well as DQN algorithm are off-policy and perform ε-greedy exploration (see Section 2.2.7). For this thesis, DQN and ACER are chosen over the state-of-the-art non-parametric Gaussian process (GP)-SARSA algorithm for policy learning [5] to allow for combined optimization of the curiosity model and the neural networks for policy learning.

2.2.6 Exploration and Exploitation Dilemma

Online decision making always balances the choice of making the best decision given the current knowledge versus gathering more information in order to be able to make better decisions in the future. It is important to gather enough information to make good decisions, but we also do not want to forgo opportunities that seem optimal given the current knowledge.

As an example: I'm baking a cake for my housemate's birthday. I could go with the

greedy option and bake the cake I always make, knowing that it will turn out well, or I could try out a new recipe. If I never try out any new recipes, I'm not exploring and might never find out that my favorite cake is actually not chocolate. In this example, the consequences of not exploring are not grave, but for an agent that has the goal to optimize birthday cakes, always baking the same one will not lead to any learning and the agent will not find the optimum recipe. On the other hand, if I do not take any of my knowledge into account, and only explore new cake recipes, the reward of tasty cakes I receive while learning might be very low, since a lot of the recipes I try contain nuts which I'm allergic too. For optimal learning, there must be a balance of exploiting the agent's current knowledge and exploration, to gain new insights and learn better ways.

The exploration versus exploitation dilemma is widely discussed and a variety of exploration techniques are used. Exploration techniques, such as Thompson sampling and Boltzmann exploration involve exploring states using uncertainty estimates. Simpler exploration methods, such as ε-greedy (see Section 2.2.7) explore by taking actions at random.

2.2.7 ε-greedy Exploration

Often, an ε-greedy policy is used to explore the environment. The ε-greedy policy choses actions at random $\varepsilon\%$ of the time, while choosing the action greedily $(1-\varepsilon)\%$ of the time. When choosing actions greedily, $a = arg\max_a Q(b,a)$ is the best action with respect to the estimated Q-function. As learning advances, the ε constant is often set to decay to a smaller value, as the agent is assumed to have explored sufficiently over time.

2.3 Intrinsic Motivation

Human learning and development often is not goal oriented, but driven by intrinsic motivation such as curiosity. Curiosity is a motivation to explore the unknown, searching for new knowledge and continuous improvement. It is visible especially in the behavior of small children, but also adults engage in many curious behaviors. Science for example often is driven by curiosity and the drive to learn more and be better rather than other expected rewards. Curious or exploratory behavior enables an agent to learn about its environment and relationship between the agent's actions, its current belief-state, and its environment. Therefore, by being curious an agent is able to gain new knowledge and skills, even when the rewards are rare or deceptive [19].

In Section 2.2.1, we say that rewards should indicate what is to be accomplished rather than how to accomplish it. This principle ensures that the agent is learning to achieve its goal and is not getting stuck in optimizing rewards for subgoals without ever reaching the overall goal. Nevertheless, there are scenarios in which formulating subgoals [8], or reformulating the overall goal can improve training and success rates. In their paper [21], Pathak et al. successfully train an agent to play *Super Mario Bros* and *VizDoom* without giving any external rewards from the game and curiosity rewards only.

For the *VizDoom* game [10], Pathak et al. consider the 3-D navigation task, where the agent's goal is to find find a vest in a series of connected rooms. They train and test their agent on different maps. It is possible for the agent to learn the game without external rewards, since the goal of finding the vest can be reformulated as exploring and learning the maps (which as a byproduct will lead to finding the vest).

For playing *Super Mario*, the conventional goal is to score points and free the princess waiting at the end of a 2-D environment, while avoiding fatal traps and enemies on the way. A curious agent receiving no rewards from the game still learns not to get killed by the enemies and to collect special effects as well as extra lives. At first glance, it seems odd that the curious agent learns those behaviors without getting any feedback for killing or dodging enemies. But since the agents goal is to maximize the intrinsic curiosity reward, it is in the agents interest to learn how to kill enemies in order to keep playing so that it can keep being curious and reach new parts of the game.

In scenarios where we are not able to formulate the overall goal in terms of curiosity maximization, such that it can be achieved by the the drive for new knowledge, curiosity still can be applied as an exploration technique in addition to external rewards. The benefit of curiosity driven learning is illustrated in Figure 2.5[4]. The picture shows a curious agent, the little chick, outperforming the goal oriented agent, the grown chicken.

[4] graphic source: T. W. Hänsch, Nobel lecture

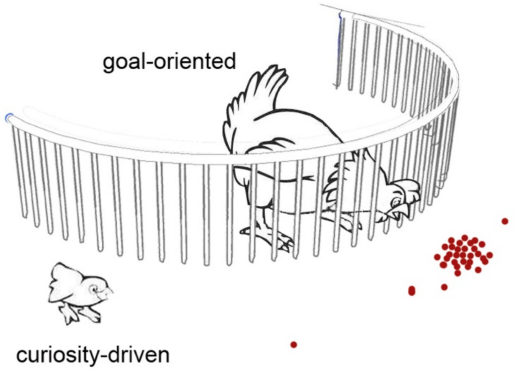


Figure 2.5 Illustrated example of the benefits of curiosity-driven learning: Curious exploring chick learning about its environment acquires more knowledge and is able to achieve higher external rewards than the goal oriented chicken that is driven by the external rewards alone.

Curiosity masters different challenges due to various reasons, but in many cases being curious succeeds simply by exploring all available options efficiently.

2.3.1 Optimal Challenge

Psychology suggests that curiosity or interest is only "engaged by what is just beyond current knowledge, neither too well known nor too far beyond what is understandable" [19]. This theory holds for curiosity-driven RL as much as for humans. A student will be unable to learn how to write a good machine learning script without first learning the basics of Python and Tensorflow. A toddler is unable to learn how to read before it understands speech and the meaning of words. For intrinsic motivated RL tasks, the concept is very similar. If no insight is gained by exploring the environment, the agent does not receive any curiosity rewards and therefore has no incentive to keep exploring. On the other hand, if the environment is well known already, for example if a student already is an expert in taking derivatives, she will not watch a suggested YouTube video titled, "How to take a simple derivate".

In RL, an agent that has perfect knowledge about all the possible states it can visit will not do anything if solely driven by curiosity rewards. The opposite case of 'too hard of a problem as for the agent to learn from curiosity' is shown by [21] when training a curious agent to play Mario Bros without any external rewards. When the agent is pre-trained on Level-1, it can quickly learn how to play the slightly advanced Level-2. Even when training for fewer iterations, the agent exhibited better results by first learning on Level-1 than skipping immediately to the more advanced Level-2. This balance of difficulty for most efficient learning is often called the "optimal challenge". In order to learn most efficiently,

the level of difficulty has to continuously increase over the course of teaching and learning [1]. In other words, curiosity rewards are only useful in an environmental setting that can be explored and learned. If things stay unpredictable for the agent (e.g. action combinations to be learned are too complex or features are random), it is too hard; if the agent is familiar with the environment and can already predict everything, it is too easy.

2.3.2 Intrinsic Motivation for SDS

In order to use curiosity rewards to drive a goal oriented dialogue system, we first have to ask the question of what we want to achieve and if it can be formulated in terms of curiosity. Then, we can think about how we want the system to act and if this can be aided by applying curious actions.

For every dialogue, the goal is to have an efficient conversation, where the agent asks questions and informs the user to successfully fulfill the user goal, as demonstrated in the example dialogue in Appendix B. Determining what the user goal is and what the values for the constraints are intuitively sound plausible to be achieved by curious actions. The question is now how the agent is learning to also take non curious actions such as informing the user? An action mask could overcome this problem, but is complex to develop in larger state-action spaces.

The main application of curiosity for dialogue policy learning should be to explore new behavior. This means that curiosity is used to explore the state-action space efficiently in order to discover optimal behavior that maximizes dialogue success.

2.3.3 Metrics for Intrinsic Motivation in Machine Learning

Intrinsic motivation has been described for machine learning tasks such as learning to play Atari games without or with spare extrinsic rewards [21], to enhance robotic motor skills [6], and drive developmental robotics [20]. In order to apply concepts of intrinsic motivation to machine learning tasks, we have to define what curiosity is and find ways to mimic its effect. As described by Oudeyer et al. [20], an intrinsic motivation system can be achieved by building

> "a mechanism which can evaluate operationally the degree of novelty, surprise, complexity, or challenge that different situations provide from the point of view of a learning robot, and then designing an associated reward. [...]

> Autonomous and active exploratory behavior can then be achieved by acting so as to reach situations which maximize this internal reward."

A variety of formulations for computing curiosity rewards have been proposed in literature, including information gain [7], entropy of actions [17], state visitation counts [3],[27], [30], and measuring errors for self supervised prediction [26], [14], [25], [20]; the methods are introduced below. These functions are, in fact, very similar, trying to either measure perceptual characteristics or measuring learning progress. The intrinsic reward signal for exploration is added as a bonus to external rewards received $r_{total} = r_{extrinsic} + r_{intrinsic}$.

Information gain about the agent's belief of environment dynamics [7] is measured using variational inference in Bayesian neural networks. In order to "maximize the reduction in uncertainty about the dynamics" Houthooft et al. [7] formalize the agent's goal as maximizing the sum of reductions in entropy $H(\cdot)$ through a sequence of actions $\{a_t\}$:

$$\sum_t (H(W|b_t, a_t) - H(W|S_{t+1}, b_t, a_t)) \tag{2.17}$$

where $w \in W$ is the parameter for the agent's model of the environment dynamics, $p(s_{t+1}|s_t, a_t; w)$, with s_t being the state at step t, b_t is the belief-state, the history of the agent up to step t, and a_t is the action taken at step t, S_{t+1} is the next state distribution. In other words, the agent is "encouraged to take actions that lead to states that are maximally informative about the dynamics of the model". The information between the next state distribution S_{t+1} and the model parameter W is written as $I(S_{t+1}; W|b_t, a_t)$, where

$$I(S_{t+1}; W|b_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathscr{P}(\cdot|b_t, a_t)} \Big[D_{KL} \big[p(w|b_t, a_t, s_{t+1}) || p(w|b_t) \big] \Big] \tag{2.18}$$

The expectation is taken over all possible next states according to true dynamics $\mathscr{P}$. The KL divergence here can be interpreted as information gain. Each term $I(S_{t+1}; W|b_t, a_t)$ at step t can be used as an intrinsic reward signal r_t^i.

Entropy of actions is a measure used to calculate mutual information and empowerment [17] and is closely related to the above described information gain. Mutual information is measuring the dependency between two random variables (RVs), empowerment $\mathscr{E}$ is the maximal mutual information I:

$$\mathscr{E}(s) = \max_w I^w(\mathbf{a}, s'|s) = \max_w \mathbb{E}_{p(s'|a,s)w(a|s)} \left[\log \left(\frac{p(\mathbf{a}, s'|s)}{w(\mathbf{a}|s) p(s'|s)} \right) \right] \tag{2.19}$$

where $\mathbf{a} = \{a_1, \ldots, a_T\}$ is a sequence of T actions leading to a final state s', $p(s'|\mathbf{a}, s)$ is the T-step transition probability of the environment, $p(\mathbf{a}, s'|s)$ is the joint distribution of action sequences, and the final state, $w(\mathbf{a}|s)$, is a distribution over T-step action sequences and $p(s', s)$ is the joint probability marginalized over the action sequence. Empowerment is a measure of the amount of information contained in the action sequence about the future state and requires internal planning. An agent maximizing the empowerment value is moving toward states from which it can reach the maximum number of future states in the planning horizon T. In order to calculate $\mathscr{E}(s)$, Mohamed at al. [17] rewrite mutual information as the difference between conditional entropies:

$$I(\mathbf{a}, s'|s) = H(\mathbf{a}|s) - H(\mathbf{a}|s', s) \tag{2.20}$$

where $H(\mathbf{a}|s)$ and $H(\mathbf{a}|s', s)$ are approximated using a variational lower bound. The intrinsic reward signal used is the approximation of empowerment $\mathscr{E}(s)$. For more detail see [17].

Count-based exploration uses state visitation counts [27], [30] with tabular RL and pseudo-counts for larger, model free environments. Bellemare et al. [3] use density models to measure uncertainty and derive pseudo-counts from the density models. This method allows to generalize count based exploration to non-tabular cases. For pseudo counts, the probability assigned to state s by the density model after observing a new occurrence of state s, $\rho'_n(s) := \rho(s; s_{1:n}s)$ is used. The pseudo-count function $\hat{N}_n(s)$ and pseudo count total $\hat{n}$ are introduced as unknowns, related through two constraints:

$$\rho_n(S) = \frac{\hat{N}_n(s)}{\hat{n}} \qquad \rho'_n = \frac{\hat{N}_n(s) + 1}{\hat{n} + 1} \tag{2.21}$$

which means that after observing an instance of s, the density model's increase in prediction of that s corresponds to a unit increase in pseudo-count. Where the pseudo-count is derived by solving:

$$\hat{N}_n(s) = \frac{\rho_n(s)(1 - \rho'_n(s))}{\rho'_n(s) - \rho_n(s)} = \hat{n}\rho_n(s) \tag{2.22}$$

Bellemare et al. [3] show how those pseudo-counts again are closely related to information gain. As intrinsic reward signal for the pseudo count method, rewards proportional to $\hat{N}_n(s)^{-\frac{1}{2}}$ are used.

The three methods introduced above all use some measure related to information gain. **Self supervised prediction errors** are used as intrinsic reward signal representing a measure of novelty of observed states. The method is to predict the next state given current state and

action, using the prediction error as indicator for novelty. Again the system gets rewarded for transitions that gain information for the agent, but the reward is not measured as information gain, using the difference between conditional entropies, but as prediction error using lack of the agents current knowledge about the state-action space. Predictions are made based on a dynamics model learned with DNNs. Prediction error as intrinsic reward signal is popular in current literature and implemented in various forms by [21], [26], [14], [25], [20], and more. For this thesis, we use a *state prediction error* reward model based on [21]'s approach. There is no particular reason prediction error is chosen over the information gain methods described above, as all of them show promising results in the gaming domain. Our method is described in detail in Section 3.1.

Using intrinsic motivation for RL addresses the exploration and exploitation dilemma (Section 2.2.6) in a new way, where exploring and exploiting become the same thing. When acting greedily, the most rewarding act for a curious agent is the one that explores most. Curious agents therefore explore efficiently and do not depend on suboptimal approaches for exploring, such as ε-greedy (Section 2.2.7).

Chapter 3

Methods

This project is inspired by state-of-the-art results intrinsic motivation yields in gaming environments and explores the use of intrinsic motivation for learning a dialogue policy. The goal is to achieve stable and sample efficient policy learning in the uncertain environment of a SDS and an environment with sparse extrinsic reward signals.

3.1 State Prediction Error as Intrinsic Reward Signal

As discussed in Section 2.3.3, there are many ways we can define curiosity. The definition depends largely on the most practical measurement to use in each specific case. Using state prediction error is a well studied approach and has been shown to work well on number of RL tasks. For *PyDial*, state prediction error is applicable using predictions of the next beliefstate. The intrinsic reward produced by this method is used in combination with DQN (Section 2.2.4) and ACER (Section 2.2.5) for policy learning.

3.1.1 Intrinsic Curiosity Module

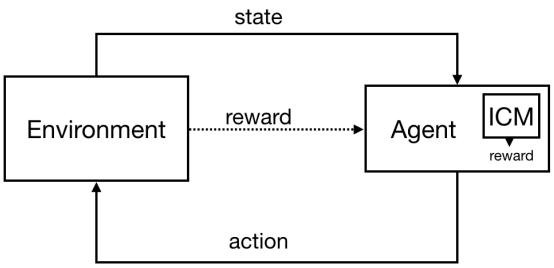


Figure 3.1 RL agent-environment interface with intrinsic motivation.

Intrinsic motivation is generated in the agent itself. Using an Intrinsic Curiosity Module (ICM), the agent can generate its own curiosity-rewards to use for RL, as is illustrated in Figure 3.1, in which there is an intrinsically generated reward that can be used with or without extrinsic reward signals coming from the environment. The ICM is part of the agent itself and performs state and action predictions. Its function is illustrated in Figure 3.2. The ICM takes action a_t, belief-state b_t, and belief-state b_{t+1} as inputs. Output of the ICM is the state prediction error, which can be used as reward for curiosity.

The ICM is adapted from Pathak et al. in [21]. This section is mirrors their method for "Self-supervised prediction for exploration". The ICM (Figure 3.2) consists of an inverse model, predicting the action a_t given states b_t and b_{t+1}, which is used to optimize the belief-state feature encoding and a forward model predicting future state $\phi(b_{t+1})$ given action a_t and state $\phi(b_t)$. The prediction error is the L^2-norm of difference between $\hat{\phi}(b_{t+1})$ and $\phi(b_{t+1})$ (see Equation 3.4). A more detailed description of both models follows below:

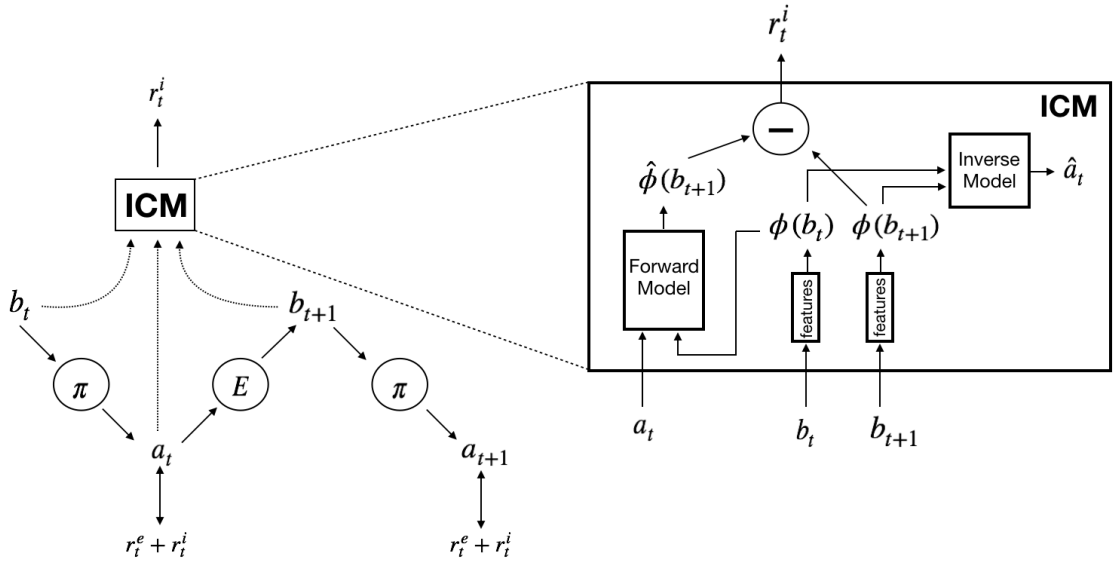


Figure 3.2 Illustrated formulation for self-supervised prediction as curiosity. In belief-state b_t the agent interacts with the environment E by executing an action a_t sampled from policy π to get to state b_{t+1}. The ICM encodes belief-states b_t and b_{t+1} into features $\phi(b_t)$ and $\phi(b_{t+1})$, that are trained to predict a_t (inverse model). a_t and $\phi(b_t)$ are inputs for the forward model predicting the feature representation $\hat{\phi}(b_{t+1})$ of b_{t+1}. The prediction error is used as intrinsic reward signal r_t^i which can be used in addition to external rewards r_t^e. (this model is adapted from [21])

Inverse Model

Predicting states in a raw sensory space corresponding to images Pathak et al. [21] use a feature representation of the original images, focusing on features that are essential in order to make good predictions. This feature space is learned by training a deep neural network with two sub-modules: the first encoding raw state b_t into a feature vector $\phi(b_t)$ and second taking $\phi(b_t)$ and $\phi(b_{t+1})$ as feature encoded inputs and predicting action a_t taken to move from state b_t to b_{t+1}. The belief-states in a SDS have no unpredictable, random features as raw images do, such as leaves blowing in the wind which could cause endless curiosity about that feature due to their unpredictability. The belief-states of a SDS are uncertain in different ways. First, the user's response might not fit the system's action and include a lot of uncertainty and randomness in itself (e.g. if the system asks "What price range are you thinking of?" a predictable response would be "I'm looking for a cheap place.", but the user might instead reply, "Actually I changed my mind I want to get Thai."). Second, even when the user reacts to a system action as predicted, for all slots there are a number of different choices with which to fill them. In order to explore the state-action space, the system does not have to learn every choice (e.g. food), which is not possible, i.e. the user can name Indian or Turkish as food type choice, in both cases a different feature in the belief-state vector (see Figure 3.3) is informed, even though the information is about the same slot: food. The system cannot learn to predict which food the user is going to request, but it can learn that after requesting information about the food a user wants, there is a high probability that the system will be informed about the food type, and one or multiple positions in the belief-state vector, belonging to food types will have non zero values. Last there is uncertainty about the user's response coming from speech recognition and spoken language understanding, and the system is not able to predict those certainty values. When using curiosity rewards for state-action space exploration, a good feature representation is more general and less detailed, e.g. a feature represents that the food type has been informed, rather than the specific type of food that has been informed. Optimizing a feature representation and reducing the size of the feature space can therefore be beneficial in order to improve the curiosity reward signal.

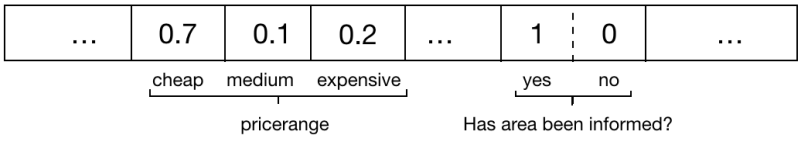


Figure 3.3 The belief-state vector representing the SDS's belief-state contains continuous fields such as for *price range* values, represented by a probability distribution, and discrete fields such as *has area been informed* that are represented by a binary value.

The function g:

$$\hat{a}_t = g(b_t, b_{t+1}; \theta_I) \tag{3.1}$$

is learned by training a NN, where $\hat{a}_t$ is the predicted estimate of action a_t and the network parameters θ_I are trained to optimize:

$$\min_{\theta_I} L_I(\hat{a}, a_t) \tag{3.2}$$

where L is the loss function measuring the discrepancy between the predicted and actual action. The output of g is a soft-max distribution across all 16 possible actions, which means minimizing L_I amounts to maximum likelihood estimation of θ under a multinomial distribution. The DNN has two sub-modules. In the first sub-module the raw states b_t and b_{t+1} are encoded into features $\phi(b_t)$ and $\phi(b_{t+1})$. The second sub-module takes the feature encodings and predicts action a_t taken by the agent to move from belief-state b_t to b_{t+1}.

Forward Model

Now a second NN is trained to predict the next state b_{t+1} in the feature space:

$$\hat{\phi}(b_{t+1}) = f(\phi(b_t), a_t; \theta_F) \tag{3.3}$$

where $\hat{\phi}(b_{t+1})$ is the predicted estimate of $\phi(b_{t+1})$. The network parameters θ_F are trained to optimize:

$$\min_{\theta_F} L_F\big(\phi(b_{t+1}), \hat{\phi}(b_{t+1})\big) = \frac{1}{2}||\hat{\phi}(b_{t+1}) - \phi(b_{t+1})||_2^2 \tag{3.4}$$

Finally, the intrinsic reward signal is the prediction error multiplied by a scaling factor η, $\eta > 0$:

$$r_t^i = \frac{\eta}{2}||\hat{\phi}(b_{t+1}) - \phi(b_{t+1})||_2^2 = \eta L_F \tag{3.5}$$

Forward and inverse dynamic losses (Equations 3.4, 3.2) are jointly optimized:

$$\min_{\theta_I, \theta_F} \Big[(1-\beta)L_I + \beta L_F\Big] \tag{3.6}$$

where $0 \leq \beta \leq 1$ is weighting the inverse model loss against the forward model loss. The ICM architecture is further discussed in Section 4.2.

3.1.2 Intrinsic Curiosity Module without feature encoding

As an alternative to the ICM described above, we describe a simpler ICM using the same principle of belief-state prediction error as intrinsic reward, but without feature encoding

and therefore without the need for an inverse model. The ICM without feature encoding is illustrated in Figure 3.4.

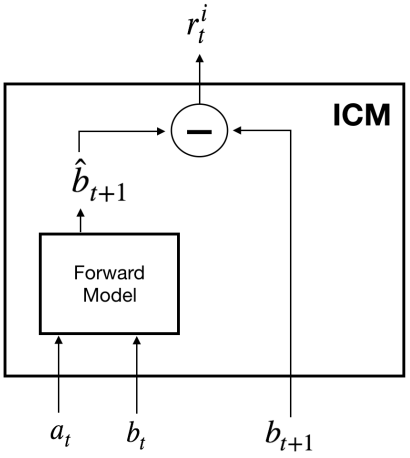


Figure 3.4 Intrinsic curiosity module without feature encoding for state prediction: action a_t and belief-state b_t are fed into the forward model predicting $\hat{b}_{t+1}$. The prediction error is used as curiosity based intrinsic reward signal r_t^i.

The **forward model** acts as before, but now is using the raw belief-sates b_t and b_{t+1} directly. Equation 3.3 for the NN now becomes:

$$\hat{b}_{t+1} = f(b_t, a_t; \theta_F) \tag{3.7}$$

The parameters θ_F are trained to optimize: $\min_{\theta_F} L_F(b_{t+1}, \hat{b}_{t+1})$.

3.1.3 Pre-trained Curiosity vs. Jointly Trained Curiosity

When jointly training the curiosity model and policy, in the early stages of training, curiosity rewards are high for all actions. In the later stages, once the model has learned, curiosity rewards will only remain for actions that prompt random/unpredictable reactions from the user. We want the curiosity reward to be used for efficient exploration, meaning to give rewards for learning the state-action space. The reward given while the ICM learns to make predictions and learning a feature space is not accurate feedback. Training the dialogue policy and ICM together means that dialogue policy learning is trying to optimize those random rewards the ICM is generating in the beginning of training, slowing down policy learning. We suggest pre-training the ICM in order to make learning of both the ICM and dialogue policy more data efficient.

For pre-training the curiosity model, data collected from running the system using different policies can be used. Running a script that is saving state and action vectors for each turn during training, a data set for pre-training is built. Since for the purpose of predicting, the actions the SDS is taking do not have to make sense, any data from training dialogues, even for less successful dialogue policies can be used to train the curiosity model.

State and action prediction are jointly trained in the same manner as when integrated into the SDS as illustrated in Figure 3.2. In each training iteration, we feed transition tuples with actions and states (a_t, b_t, b_{t+1}) as batches into the NN, minimizing forward and inverse loss (Equation 3.6).

Chapter 4

Experiments

In this chapter the performance of the intrinsic motivation method for exploration as described previously is evaluated through a number of experiments. The baseline is an ε-greedy policy.

First, settings of the SDS and ICM are introduced and baseline results are presented. Next a number of experiments to investigate exploration and curiosity are conducted and settings for best ICM performance are explored.

4.1 PyDial Experimental Setup

The statistical spoken dialogue system used for this research is *PyDial* [31], developed for scientific purposes by the Dialogue Systems Group at Cambridge University. *PyDial* is a Python open-source statistical spoken dialogue system toolkit. The toolkit provides domain independent implementations of all the dialogue system modules as well as simulated user and error models. Different benchmark environments and models for the dialogue manager are provided in the toolkit.

In the experiments conducted for this thesis, the Cambridge Restaurants (CR) domain is used, where, in a goal oriented dialogue, the user can access information about restaurants in Cambridge, UK. The action space for this domain is described further in Appendix A. We train the dialogue policies with the DQN and ACER method (Sections 2.2.4, 2.2.5), which are provided as options within *PyDial*. We use the focus belief tracker described in Section 2.1.3 and the user simulator described in Section 2.1.5. To allow for the best comparison the policy models (DQN & ACER) use both the same settings in all experiments, with seizes of their hidden layers set to $h_1 = 300$ and $h_2 = 100$. Training steps are performed with minibatches of 64 dialogues and the experience replay memory has a capacity of 6000 dialogues. If

not specified otherwise results are given for testing iterations, which are performed after a number of training iterations. We run training in 200 and 1000 iteration intervals, evaluating the partially trained policies with 200 and 500 iterations of testing. One iteration is the running of one full simulated dialogue. The total number of training for all experiments is 4000 dialogues. All results are averaged over 5 runs, using different random seeds, in order to reduce variance coming from different random initializations. Where ε-greedy exploration (Section 2.2.7) is used in training, testing is performed following a greedy policy without exploration. Where curiosity is used instead of ε-greedy exploration, the curiosity reward is included in training as well as testing[1]. Any dialogue is limited to 25 turns after which it is terminated as unsuccessful, this is in training and testing. Four different environments are used, with semantic error rates of 0% and 15% and with or without execution mask for action selection as shown in Table 4.2. For all experiments external rewards received are a success reward of +20 at the end of a successful dialogue and turn penalties (negative reward) of -1 as feedback for every turn.

4.2 Intrinsic Curiosity Module - Experimental Settings

The ICM consists out of two NNs, an inverse and forward model, for action and belief-state prediction. First, the inverse model maps the input belief-state vectors b_t and b_{t+1}, of size 268, into feature vectors $\phi(b_t)$ and $\phi(b_{t+1})$ by passing them as inputs into a fully connected layer of l_1 units and an exponential linear unit (ELU) as activation function. Next, $\phi(b_t)$ and $\phi(b_{t+1})$ are concatenated into a single feature vector and passed as input, size $2 \times l_1$ into a fully connected layer of l_2 units followed by an output fully connected layer with 16 units to predict one of the 16 possible summary actions. Now, for the forward model, $\phi(b_t)$ and a_t are concatenated into a single vector of size $l_1 + 16$ to be passed into a sequence of two fully connected layers with l_2 and l_1 units. Sizes used for the layers are specified by the variables l_1 and l_2, their values are displayed in Table 4.1. In the following sections to l_1 is referred to as feature size. Equation 3.6 is minimized with learning rate 0.001, β is 0.2.

Some of the experiments use the ICM without an inverse model and feature encoding. For this the forward model is used alone. Belief-state b_t and action a_t are inputs for the forward model directly, which predicts the next belief-state $\hat{b}_{t+1}$.

[1] Curiosity rewards do not affect the policy during testing, since the behavioral policy is fixed.

feature size	layer 1 (l_1)	layer 2 (l_2)
20	20	77
77	77	77
120	120	200
200	200	200
full (268)	268	268

Table 4.1 Settings for different ICM architectures used.

4.3 Baseline

	Environment 1	Environment 2	Environment 3	Environment 4
SER (%)	0	0	15	15
Masks	On	Off	On	Off

Table 4.2 SER model and action mask environments for the experiments.

As a baseline for all experiments, the setting with rewards of $r_{success} = +20$ for successfully completed dialogues and $r_{turn} = -1$ as turn penalty is used. Therefore, the accumulated reward received for a full dialogue is $r = 20 - n$, where n is the number of turns in a dialogue. The baseline uses ε-greedy exploration with the ε value starting at 0.3, linearly decreasing to zero over all dialogues. The main setting for our experiments is environment 4, which uses a 15% error rate and no action selection mask. A summary description for all environments used is shown in Table 4.2. Baseline results for all environments are shown in Figure 4.1 for ACER and in Figure 4.2 for DQN. All average rewards and success rates displayed in this thesis are plotted with 95% confidence interval (CI) error bars. For ACER, using the action mask (environment 1 & 3) results in more data efficient policy learning and higher average success rates are achieved than for the corresponding SER without action selection masks (environments 2 & 4). SER of 15% compared to SER = 0% results in nearly a 5% drop in the final dialogue success rate. DQN baseline results show the same trend, but demonstrate DQN as being a more unstable learning method with higher variances and inconsistent performances inconsistent. In environment 1, the easiest environment performance even decreases when training advances (Figure 4.2). Furthermore DQN learning appears to be less data efficient than ACER and to achieve lower final success rates.

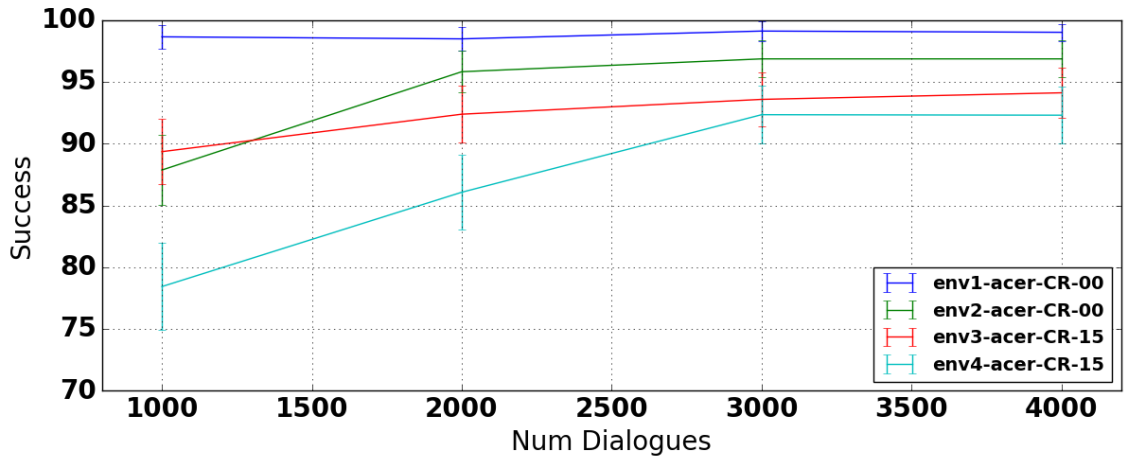


Figure 4.1 Average reward and success rates for testing the 4 baseline environment settings after every 1000 dialogues of ACER policy learning.

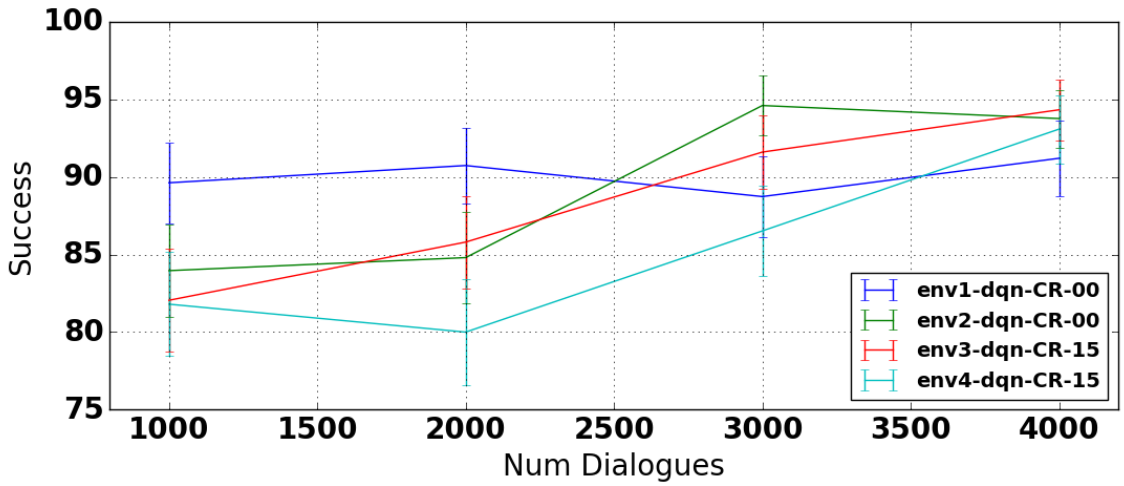


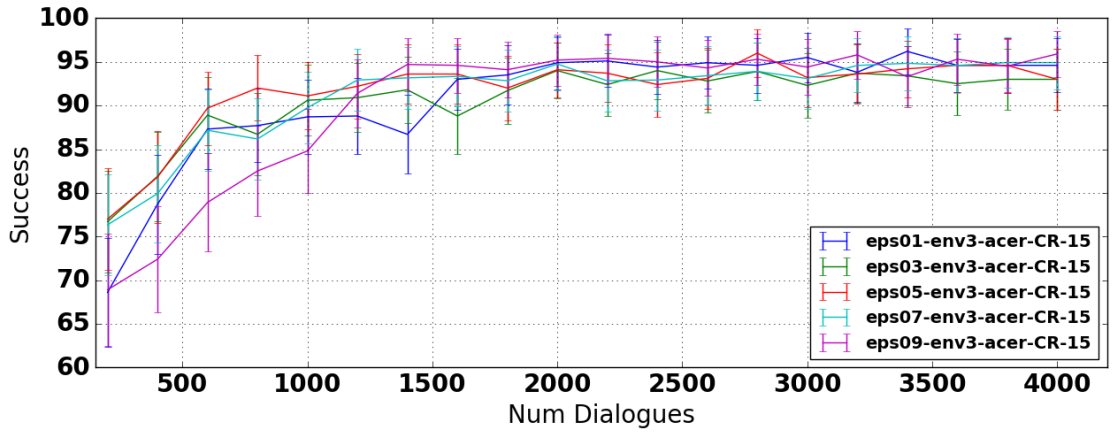
Figure 4.2 Average reward and success rates for testing the 4 baseline environment settings after every 1000 dialogues of DQN policy learning.

Results in the following sections are only displayed for for the more stable and better performing ACER.

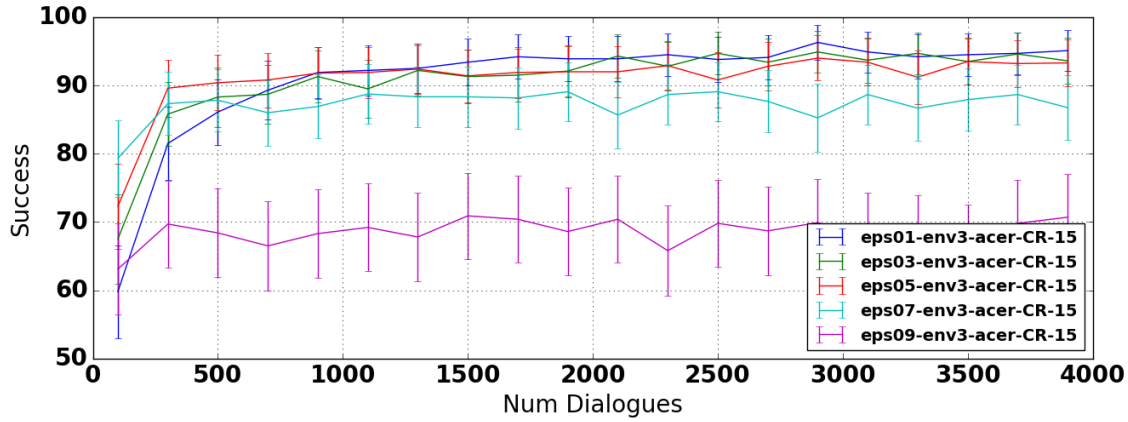
4.4 ε-greedy Exploration

In order to investigate the benefits of increased and more efficient exploration for SDS policy learning, we start with an investigation of the ε-greedy exploration, using different ε settings. In all cases the ε value is linearly decreased to 0 over all training iterations. Average success

rates for testing are shown in Figure 4.3a. Average in training performance is shown in Figure 4.3b.



(a) Average testing success rates.



(b) Average training success rates.

Figure 4.3 Performance for varying ε settings for ε-greedy exploration in environment 3 using ACER. Displayed are testing (a) and in-training results (b).

Testing (Figure 4.3a) of the learned policies shows, that very large ($\varepsilon = 0.9$) as well as very little ($\varepsilon = 0.1$) exploration leads to less data efficient policy learning than using less extreme ε values ($\varepsilon = 0.3 - 0.7$). This is since an agent mainly acting at random will need more data to randomly discover optimal trajectories, and an agent not exploring much will need more data to learn new, better trajectories. Section 2.2.6 discusses this exploration exploitation dilemma and the need to find a balance of exploration and exploitation.

After 4000 dialogues, final policy performance for all ε settings is about equal. The in-

training performances (Figure 4.3b) are however much lower for higher ε (0.7 and 0.9) due to the high randomness in action selection.

4.5 Curious Actions

Since intrinsic motivation can be used to foster curiosity about both user goals and exploration of the state-action space, we investigate the summary actions from which the system earns curiosity rewards. This is to learn which actions the system is taking in order to earn curiosity rewards. Figure 4.4 displays the last 500 curiosity rewards received with their corresponding actions during training for 3000 iterations. This experiment is run with pre-trained ICM without inverse model and η, the reward scaling factor is set to 100. Environment 3 is used, meaning that the actions are masked. We use a masked environment here in order to show curiosity rewards for a larger number of actions.

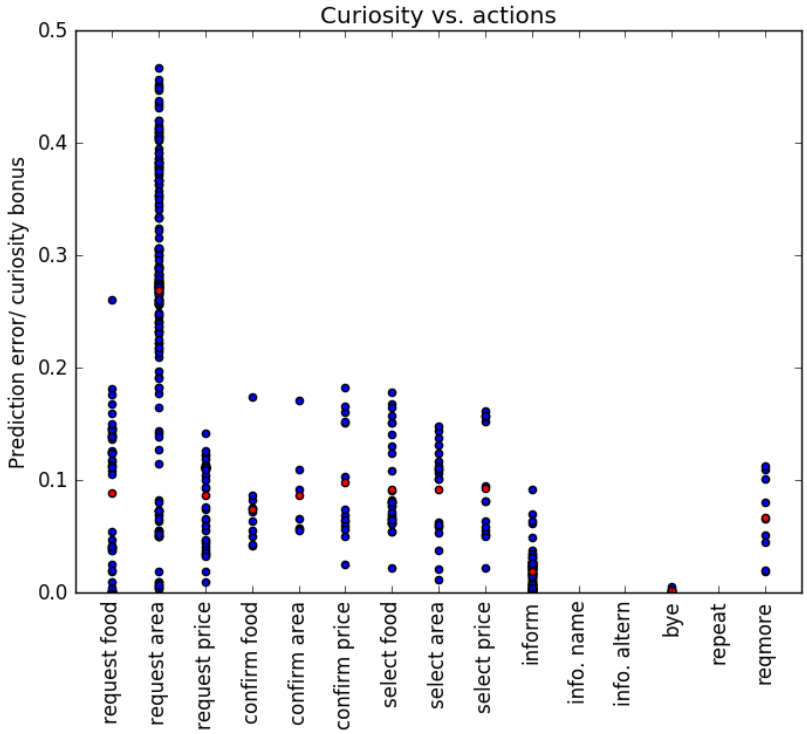
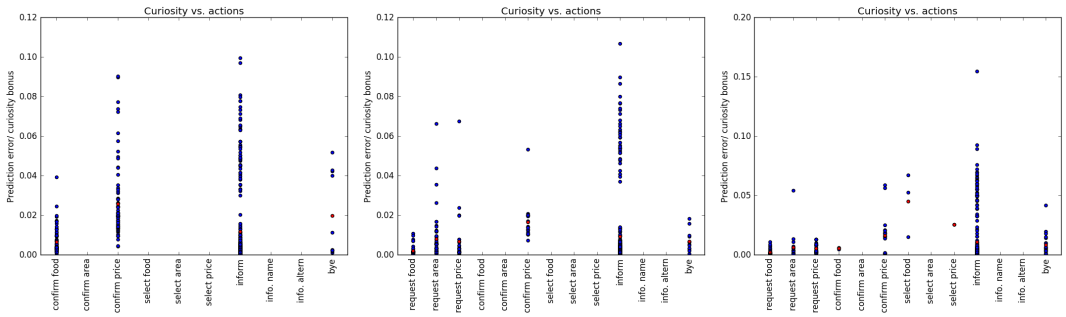


Figure 4.4 Scatter plot of all actions taken and corresponding curiosity bonus received during training, average rewards for each action in red.

The algorithm learns to use most summary actions, the `inform_byname()` and `inform_alternative()` as well as the `repeat()` actions are not used at all. As expected, the system can successfully predict that the dialogue is over, after using action `bye()`. Informing the the user mostly leads

to very predictable state changes, indicated by the low average curiosity reward earned. Action `request_area()` earns by far the highest curiosity rewards out of all actions. Predicting the state after the user informs the system of an area might have more error cases, than when predicting after the system requests food or price, since price only has three options and the food slot is most likely to be informed by the user without the system having to request it first.

Figure 4.5 shows how the policy learns to use more actions during the course of training. With more efficient exploration, the new, and potentially better suited, actions are learned faster.



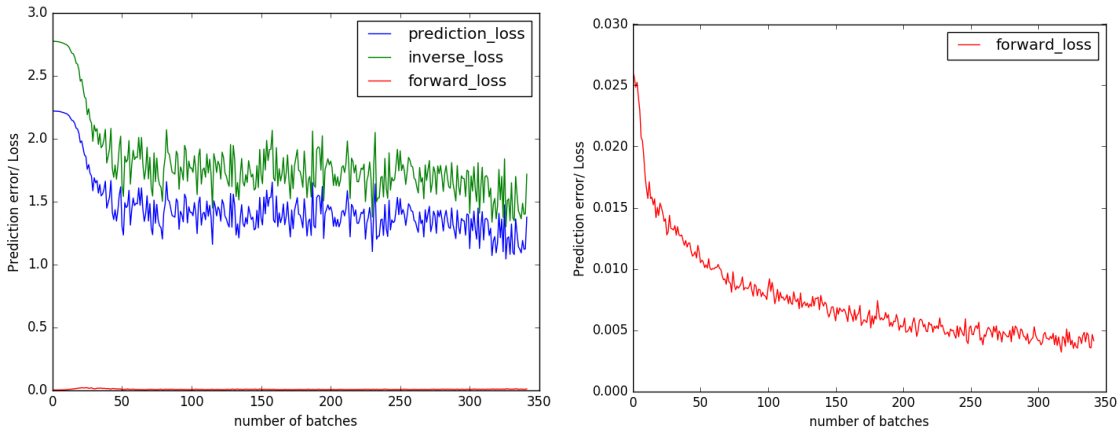
(a) 200 dialogues (b) 400 dialogues (c) 600 dialogues

Figure 4.5 Actions the policy has learned to use after training for 200, 400, and 600 dialogues and corresponding curiosity rewards those actions received. This shows the evolution policy's learning.

4.6 Pre-training

Learning a prediction model in the uncertain environment of a SDS turns out to be too challenging for the NN, when done in parallel with policy learning, where the prediction errors are used as rewards. The model is not learning and losses remain constant.

For pre-training, we use data collected during ACER policy training with baseline environment 1 (see Table 4.2). Transitions (b_t, a_t, b_{t+1}) for dialogue 2000-3000 of dialogue training are saved into a pre-training data pool, from which we sample random minibatches (size: 64 transactions). We train the model for 340 iterations.



(a) Feature size =20.

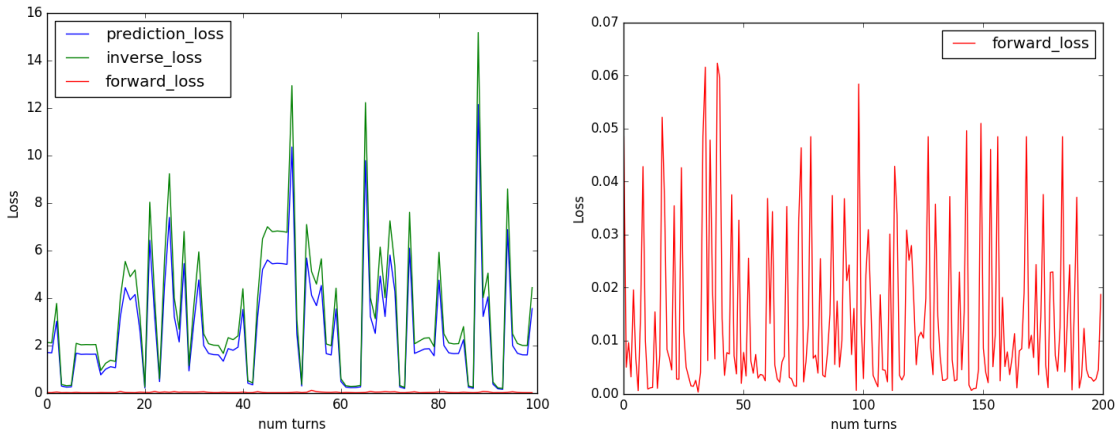
(b) No feature encoding.

Figure 4.6 Losses during ICM pre-training for the ICM with and without inverse model.

Inverse, forward and combined prediction loss during pre-training are shown in Figure 4.6a. The optimized combined prediction loss is the linear combination of forward and inverse loss (Equation 3.6). Figure 4.6b shows the forward loss during pre-training the ICM without inverse model.

When using pre-training for the curious learning agent, the ICM does not have to learn feature encodings and prediction models from scratch in an environment with noisy curiosity rewards. This procedure is comparable to Pathak et al. [21] training their ICM when playing *Super Mario Bros* in level-1, and then using that model to achieve better results in level-2, since starting ICM training in level-2 is very data inefficient. The characteristic of improved learning through optimized difficulty levels is described in more detail in Section 2.3.1.

Figure 4.7a shows inverse, forward and combined prediction loss during policy learning. The forward loss, which is the belief-state prediction error, used as intrinsic reward signal (Figure 4.7b), is much smaller in magnitude than the inverse loss.



(a) Forward, inverse and combined loss.

(b) Forward loss (intrinsic reward signal).

Figure 4.7 Losses during policy learning with pre-trained ICM.

4.7 Scaling the Curiosity Reward Signal

The intrinsic reward signal has to be proportional to the turn penalty and success reward. External reward signals +20 for success and -1 as turn penalties are applied as before. The curiosity reward is scaled to be in an appropriate ratio to those external rewards. When the curiosity reward given is too big, the system does not learn a policy that is maximizing successful dialogues, but instead its main goal becomes to maximize curious actions. The goal in this section is to find an appropriate setting for scaling factor η, that results in reward and success curves moving in parallel. The experiments in this section are conducted in environment 1, which has the simplest settings, using an action mask and SER of 0%.

First, we set the reward scaling factor $\eta = 100$, to effectively even out the turn penalty. Rewards and success rate for this are shown in Figure 4.8. It is apparent that $\eta = 100$ is too large, the reward received is growing over time, but no successful dialogues are achieved.

Average success rate and rewards for a curious agent with $\eta = 10$ are shown in Figure 4.9. $\eta = 10$ allows the learning of a policy that leads to successful dialogues, but as learning advances, the maximized reward grows, the success rate decreases. For $\eta = 1$, represented by the green line in Figure 4.10, average reward and success rate grow together, suggesting that optimizing received rewards correlates to optimizing the dialogue success rate. Using $\eta = 0.1$, represented by the blue line, yields the highest success rates out of the 4 values

considered. The red line presents a greedy policy without exploration[2], i.e. $\eta = 0$. With this line, we can compare that the curiosity signal to an agent without exploration and see if the curiosity reward is too small to impact the policy.

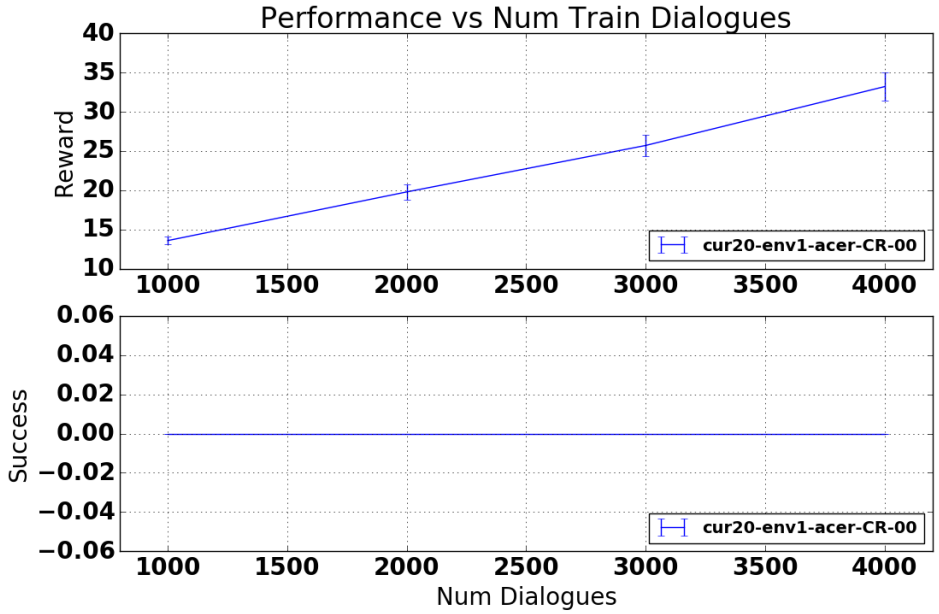


Figure 4.8 Policy learning performance for $\eta = 100$.

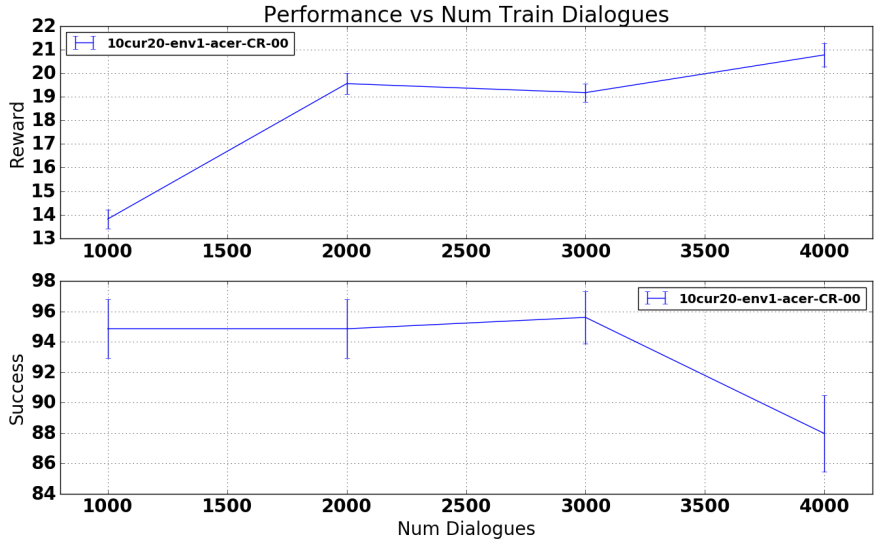


Figure 4.9 Policy learning performance for $\eta = 10$.

[2]Note that the greedy policy without exploration yields very high performance because of the action mask used in environment 1.

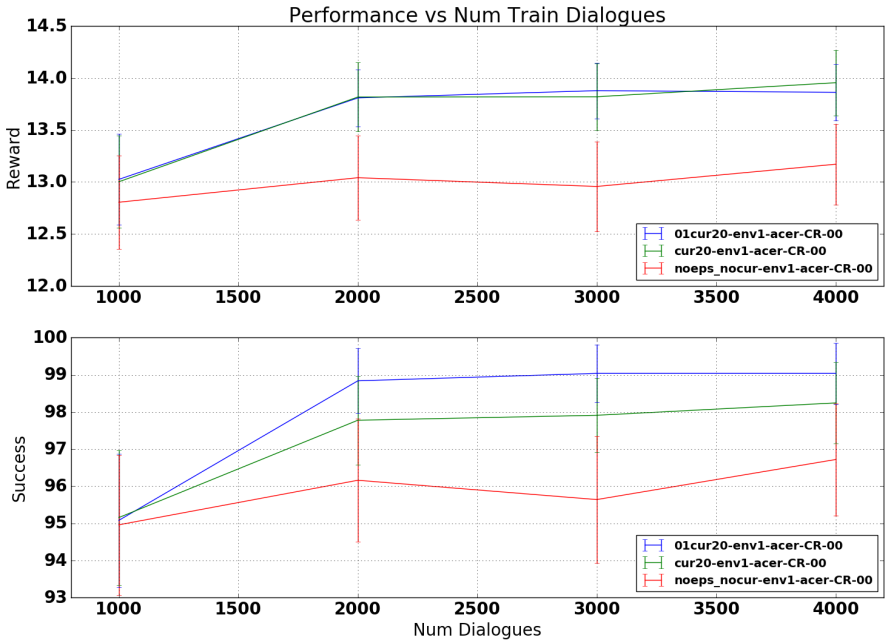


Figure 4.10 Policy learning performance for $\eta = 1$ and $\eta = 0.1$, with $\eta = 0$, a sole greedily acting policy as baseline in red.

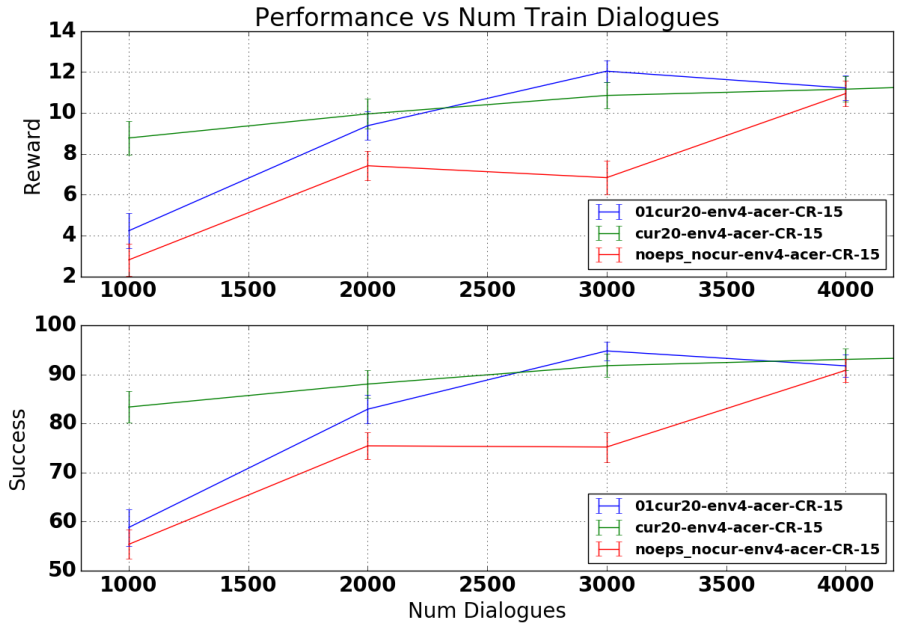


Figure 4.11 Average reward and success rate for $\eta = 0.1$, in blue, and $\eta = 1$, in green, for environment 4, no exploration in red.

In Figure 4.11, average reward and success rates for $\eta = 0.1$, in blue and $\eta = 1$, in green, are shown for the more complex environment 4. Policy learning with the lower $\eta = 0.1$ for reward scaling is less data efficient than when the higher curiosity rewards, $\eta = 1$ is used. Again, the red line shows a benchmark policy using no exploration. Final average success rates after 4000 training dialogues are 90.8%, 91.8%, and 94.2% for no exploration, $\eta = 0.1$ and $\eta = 1$ respectively. Those results indicate that in a more uncertain environment (environment 4), larger curiosity reward signal are needed than in less noisy environments (environment1).

4.8 Tuning of Belief-State Feature Encoding

A good feature space for the curiosity model should only represent features that the agent can learn to predict given belief-state b_t and action a_t. Any other information encoded adds noise to the prediction error. Instead of hand-designing this feature representation of the belief-state, it is learned with a NN. Weights for the feature encoding NN and the forward prediction model are trained simultaneously with the policy training. For those experiments, environment 4 and reward scaling factor $\eta = 1$ are used.

We try a range of different feature sizes 20, 77, 120, 200 and the full size of the belief-state vector, 268[3]. ICM settings for the different feature sizes are shown in Table 4.1. Average rewards and success rates for the use of different feature sizes are illustrated in Figure 4.12 where `cur-full` corresponds to the full belief-state, with no feature encoding being used, `cur20` to `cur200` are the curiosity configurations with feature sizes 20 to 200, and env4-acer-CR-15 is the baseline setting.

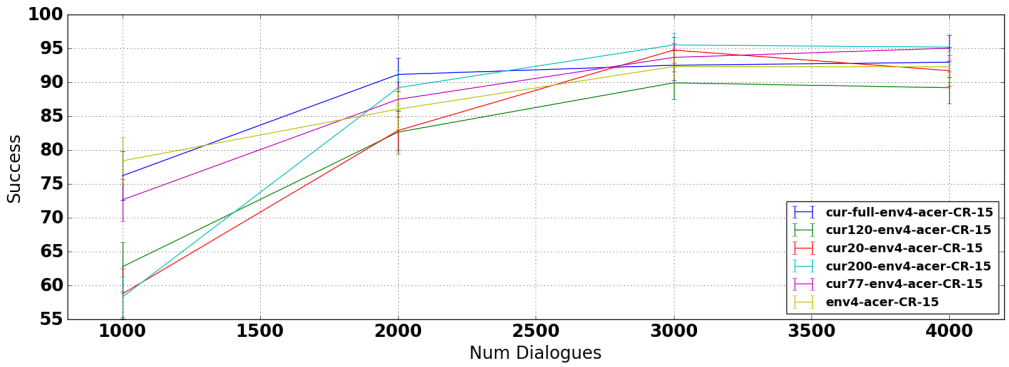


Figure 4.12 Different feature encoding settings for the ICM. Performance during policy learning with ACER, environment 4.

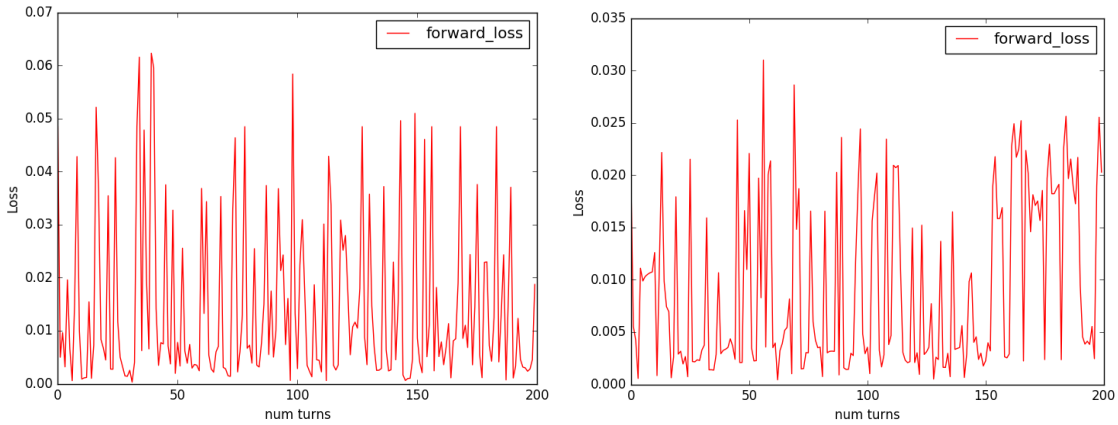
[3] no feature encoding

Belief-state prediction without feature encoding is shown to be more data efficient than the models using feature encoding. `cur77` is the most data efficient feature encoding model and also the one with the highest final success rate of 95%, together with `cur200`. `cur200`, `cur77`, and `cur-full` achieve a higher final average success rate than the baseline (92.3%) and `cur20`'s performance is just below the baseline. `cur120` has the lowest average success rate with a final rate of 89.5% the only example to fall below the 90% mark. Using no feature encoding is more data efficient since the ICM without the inverse model is much less complex and easier to learn. Feature encoding is shown to be beneficial later in training, achieving higher success rates than the ICM with forward model only. An explanation for the benefit of feature encoding in the long run could be that the feature encoding learns more general features for information about user specific choices (e.g. the feature for food type after feature encoding might not be a slot for each food (Italian, Indian, Chinese, ...), but a slot for if the food has been informed and with how certain the system is about which specific food type the user wants without having to state the actual type.) No user specific information is needed, since this information is user dependent (not learnable), thus invoking unnecessary curiosity for a random feature.

4.9 Curiosity Without Feature Encoding

As already observed in the previous section, it is not necessary to use feature encoding for the belief-state prediction error as a curiosity reward signal. This does not necessarily indicate that all features are important for making predictions, but implies that there are no features that are invoking misleading curiosity (e.g. random movements of leaves blowing in the wind in images).

During the earlier stages of learning, a curiosity model with no feature encoding achieves a higher average success rate than a model with feature encoding. In the previous section we mentioned that this can be due to the higher model complexity when forward and inverse model are used in the ICM. Figure 4.13 shows the forward losses for both ICM, with and without inverse model, the spikes in Figure 4.13a are much greater in magnitude than for Figure 4.13b. The greater variance of the forward loss, which is used as reward signal, for the ICM with feature encoding, reflects its instability and is the reason for less data efficient policy learning than when the less complex model with no feature encoding is used.



(a) Feature size = 77.

(b) No feature encoding.

Figure 4.13 Forward loss (curiosity reward) during policy training with ICM with and without inverse model.

Average success rate plots in 4.14 show that feature encoding for the belief-states is not crucial for learning curiosity as it is for images [21] (again, this is since images encode more information than only that which is in the agents control and what is controlling the agent). The baseline, exploring without a curiosity model is most data efficient, but has the lowest final average success rate. Policy learning with the feature encoding ICM shows less data efficient learning, but the highest final average success rate, while policy learning without feature encoding is more data efficient than with feature encoding, but only results in a marginally better average success rate than the baseline.

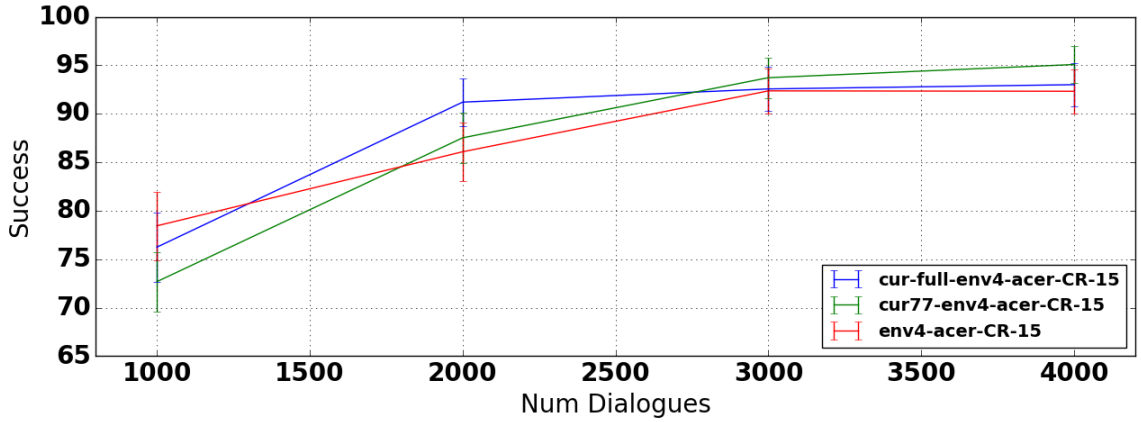


Figure 4.14 Average success rates for models exploring curious with and without feature encoding in environment 4.

In images certain unpredictable features can evoke endless curiosity and distract from features that the agent actually can influence or is influenced by. The nature of the dialogue belief-state

is that it already only contains information necessary to the DM and for decision making. Nevertheless, not all information that is needed for the DM is also needed for curiosity. Having an inverse model and learning a belief-state feature encoding is more complex, but can lead to higher final success rates when used in dialogue policy learning.

4.10 ICM + ε-greedy

Using curiosity rewards we achieve higher final average success rates than with the ε-greedy baseline method, but the baseline has the highest average success rate when evaluating the policy after only 1000 iterations (see Figure 4.14). In this experiment, both methods are combined, random ε-greedy exploration is used in addition to the curiosity reward for exploration. For feature encoding, feature size 200 settings and $\eta = 1$ are used. This configuration achieves 84.2% average success rate after 1000 training dialogues, by over 5% the highest success rate, thereby being the most data efficient out of the methods displayed. With a final average success rate of 93% after 4000 dialogues, it performs better than the baseline but 2% below the performance of curiosity rewards for exploration alone.

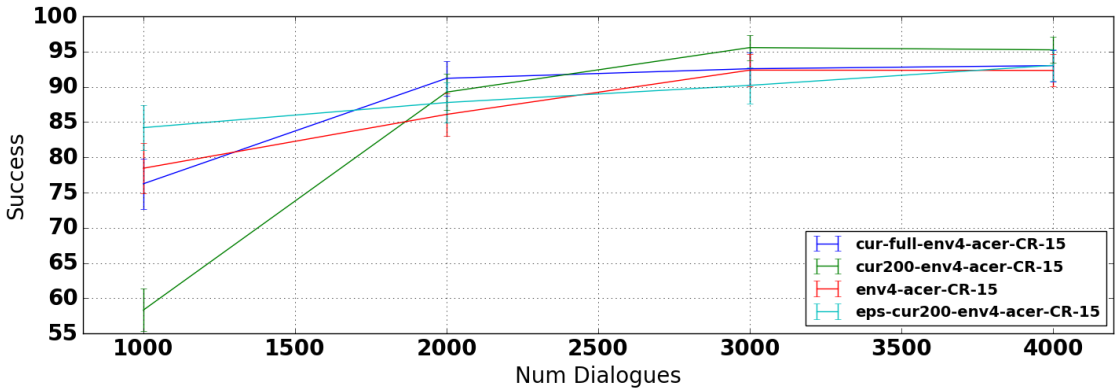


Figure 4.15 Average success rates; including ε-greedy + curiosity rewards in light blue.

4.11 ICM Performance on Predicting Dialogue Belief-States

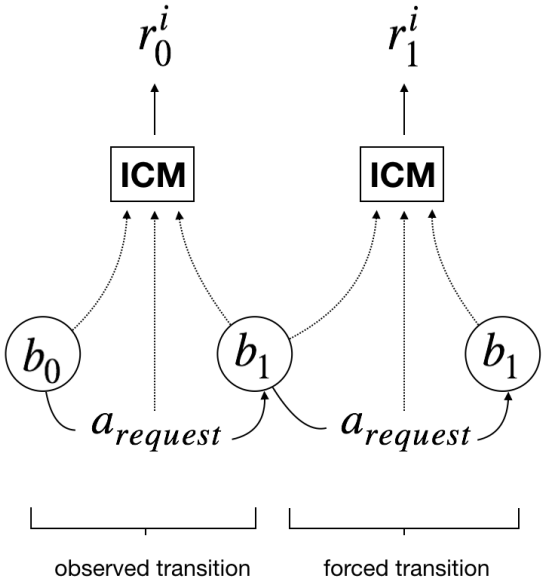


Figure 4.16 State transitions and intrinsic reward model.

In order to validate that the curiosity module is working and making good predictions, we test the ICM by feeding chosen belief-states and actions into the pre-trained model (with feature encoding). This test is based on the assumption, that the ICM should be able to predict that the belief-state is not changing much after twice repeating the same `request` action. The procedure is illustrated in Figure 4.16. We feed transition tuple (b_0, $a_{request}$, b_1), into the ICM to receive r_0^i, the prediction error, as a reward, where the transition tuple (b_0, $a_{request}$, b_1) is sampled from dialogue learning in a SER $= 15\%$ environment. Next we feed transition tuple (b_1, $a_{request}$, b_1) into the ICM. For this experiment, we assume the user's response and systems recognition to be so that the next belief-state equals the first. We want the ICM to predict that the belief-state, after repeating the same `request` action twice in a row, has a similar feature encoding to its state before performing the action. Results (see Table 4.3) show that the prediction error is reduced by a significant amount, indicating that the ICM is learning to perform accurate belief-state predictions. Note that we assume a user response matches the system's request action. In reality, there is uncertainty about the response, even with a user simulator, and the user might decide to present the system with different information than requested.

r_0^i	r_1^i
0.043	0.001
0.052	0.001

Table 4.3 Experimental Results for curiosity rewards, rounded to 3 decimal places.

4.12 Summary

In this chapter, settings for the use of an ICM generating curiosity rewards in *PyDial*'s DM are investigated and evaluated, using ACER dialogue policy learning.

Experimentally, we establish the need of ICM pre-training, the setting of reward scaling factor $\eta = 1$ when used in environment 4, and appropriate feature sizes for the belief-sate vector used in *PyDial*'s CR domain. Feature encoding proves to be valuable, but not crucial to the task, as it can be in domains such as image recognition. In Table 4.4, results are shown for different feature encoding settings. Using no feature encoding has the lowest final average success rate of 93%, still beating the baseline of 92.3%. Feature sizes 77 and 200 are shown to work well for this domain with final average success rates of 95.1% and 95.2% respectively. Highest average reward and lowest number of turns are achieved with feature size 77. For final evaluation of our model we, chose environment 4, since it is the one most realistic to when real users are to be used.

	Baseline	No feat. enc.	Feat. size 20	Feat. size 77	Feat. size 200
Reward	11.3	11.0	11.2	**12.2**	11.8
Success rate (%)	92.3	93.0	93.1	95.1	**95.2**
Turns	7.2	7.6	7.5	7.0	7.3

Table 4.4 Average reward and success rates after 4000 training dialogues, excluding ICM pre-training, environment 4.

Note that rewards for policy learning with turn penalty and success reward only do not compare to rewards for policy learning when the ICM is used to generate additional intrinsic rewards. In Table 4.5, we compare the best results achieved in this work (ACER with ICM) to DQN and GP-SARSA [5] results in policy learning within *PyDial*, using the same CR domain and environment. As before, all results are averaged over 5 random seeds.

	4000 training dialogues			1000 pre-trg +3000 trg	
	GP-SARSA	DQN	Baseline (ACER)	Feat. size 77	Feat. size 200
Reward	9.8	11.5	11.3	11.7	12.0
Success rate (%)	90.8	93.1	92.3	93.7	**95.6**
Turns	8.4	7.2	7.2	7.2	7.1

Table 4.5 Average reward and success rates after 4000 training dialogues, environment 4.

The ACER with ICM is trained with 3000 dialogues before testing, in order to account for the 1000 dialogues required for ICM pre-training. ACER with ICM model, feature size 200, achieves an average success rate of 95.6% after 3000 dialogues of policy training. With a p-value < 0.002 the difference to any of the three models without intrinsic curiosity rewards is statistically significant. Training a smaller feature size of 77 takes more dialogues, and the final success rate is 93.7% after 3000 training iterations, which is only statistically significantly different to the GP-SARSA result with p-value $= 0.008$, but not to its ACER baseline or DQN. Out of the three policy models without curiosity rewards, DQN scores the highest average reward and a success rate of 93.1%. GP-SARSA scores lowest with an average success rate of 90.8%.

Chapter 5

Conclusion and Future Work

5.1 Thesis Summary

SDSs are suspect to a lot of uncertainty coming from speech recognition and spoken language understanding as well from users themselves. Further, SDSs operate with large state-action spaces, making dialogue policy learning a hard problem. The RL based dialogue manager learns a policy by maximizing rewards received. The aim is to implement a reward system that deals with the uncertain environment of the SDS and sparse user feedback. In this thesis we use ACER for dialogue policy learning, with intrinsic generated rewards to guide exploration. Belief-state prediction error is used as the intrinsic reward signal. The signal is generated with an intrinsic curiosity module (ICM) inspired by Pathak et al. [21]'s ICM for intrinsic rewards in gaming. Experiments show that belief-state prediction error as curiosity reward signal can replace ε-greedy exploration in DRL for SDS. A number of settings for the ICM are evaluated. First, we discover that the ICM is not able to learn a prediction model when trained in parallel with the policy. Pre-training the ICM with about 1000 dialogues and potentially adapting it to the small data pool helps to learn forward and inverse prediction models and belief-state feature encoding. The pre-training allows us to successfully train the ICM later on in parallel with the policy. For best success rate results, the prediction error reward signal has to be scaled to be in an appropriate ratio with success rewards and turn penalties used. We find that in a more complex environment, larger rewards help to be data efficient, while in an environment with low uncertainty a smaller curiosity reward signal is sufficient and can result in higher success rates than when using a larger signal. If the curiosity reward signal grows too large, it interferes with the success reward received and maximizing rewards does not lead to improving success rates anymore, but leads to purely maximizing exploration instead. Belief-state predictions are performed within a feature space and learned by a self-supervised inverse model. Feature encoding is performed to improve predictions

by letting the inverse model learn a feature space that optimizes inverse predictions. Ideally, the model learns a belief-state feature encoding that is robust to uncertainties coming from the various units of the SDS and the user itself, making predictions only dependent on the most important factors. Out of four different settings, we find feature sizes of 77 and 200 to work best in representing our belief-state vector of size 268. With a feature size of 200 and reward scaling factor $\eta = 1$ we achieve a 3.3% increase in success rate over the ACER baseline. We further investigate belief-state prediction error without using feature encoding as curiosity reward signal. Results show that the less complex prediction model is data efficient, learning a successful policy after fewer training iterations, but cannot achieve the same final performance as when feature encoding is used. Fastest learning is achieved by combining ε-greedy and curious exploration, but is showing lower final success rates than exploring purely curious.

5.2 Discussion and Future Work

This project shows that using curiosity rewards for exploring the state space in dialogue policy learning with ACER can outperform simple ε-greedy dialogue policy learning. Using curiosity rewards for exploration we achieve success rate results better than ACER, DQN, and GP-SARSA with their current exploration methods.

Curios exploration is more efficient than random exploration and should enable exploration of large state-action spaces more data efficient (given that the model is able to learn a good feature encoding) and yield higher success rates faster. Curiosity has the potential to work well in very large state-action spaces where the model benefits from learning generalizations through feature encoding and efficient exploration is crucial to success.

5.2.1 Improving Current ICM Architecture

The use of belief-state feature encoding for predictions is evaluated and found to be beneficial in further improving the curiosity signal when using prediction error as curiosity rewards. Even though few settings for the prediction model are tested and evaluated, they already show promising results. More work can be done to improve the inverse and forward model NNs for belief-state prediction and feature encoding in order to increase success rates and data efficiency. The reward scaling factor can be further tuned to better match different feature sizes of the encoding.

5.2.2 ICM Within Different Policies

In this work a DNN model is used for policy learning allowing the ICM and policy to be jointly optimized. We decide not to use that option, which means that the ICM as introduced can also be used with any non-DRL RL policy learning model. For future work, we recommend to evaluate the performance of using the ICM's intrinsic reward signal in place of other exploration techniques currently used in dialogue policy learning and testing how joint policy and ICM optimization affects performance within DQN and ACER.

5.2.3 Real Users

As curiosity rewards are shown to work well with simulated users, the next step is to use curiosity for policy learning with real users. Curiosity for exploration is more promising than simple ε-greedy policies, since it is not dependent on random actions in training and explores the state-action space efficiently. How well curiosity rewards work in an environment with real users has yet to be shown.

5.2.4 Pre-training

In order to solve the pre-training problem in a real user environment without access to labeled data, instead of using labeled data, training can be performed starting out with an ε-greedy policy using data from those first dialogues to train the ICM. Pre-training can also be performed with a simulated user, leading the system in the right direction before adjusting it to real user interactions. Experiments show that pre-training is beneficial to train a feature encoding that filters out irrelevant information provided in the original feature vectors. Having a good feature encoding then enables the system to make accurate predictions and further improve the feature encoding when gathering new information. The ICM is not meant to learn much about the state-action space before being used for policy learning, which would defeat its goal. When pre-training a curious agent, we must remember, that we want the agent to be curious and we therefore should not feed it valuable information that we want it to discover and learn for the policy later on, but only feed it enough information to allow it to learn a simple feature encoding as a staring point. Pre-training is making dialogue policy learning with curiosity rewards more data efficient by using better suited difficulty levels for learning (see Section 2.3.1). When pre-training the ICM on a small set of dialogue transition tuples, it learns faster than if used with a greater variety of real transitions for policy learning directly, where it mostly fails to learn anything for the complete set of 4000 training iterations.

References

[1] Ahmed, S. (2017). Theory of sustained optimal challenge in teaching and learning. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 61(1):407–411.

[2] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning.

[3] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation.

[4] Fatemi, M., Asri, L. E., Schulz, H., He, J., and Suleman, K. (2016). Policy networks with two-stage training for dialogue systems.

[5] Gasic, M. and Young, S. (2014). Gaussian processes for POMDP-based dialogue manager optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(1):28–40.

[6] Hafez, M. B., Weber, C., and Wermter, S. (2017). Curiosity-driven exploration enhances motor skills of continuous actor-critic learner.

[7] Houthooft, R., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. (2016). Vime: Variational information maximizing exploration.

[8] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2). Reinforcement learning with unsupervised auxiliary tasks.

[9] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99 – 134.

[10] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). Vizdoom: A doom-based ai research platform for visual reinforcement learning.

[11] Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. (2016). Deep successor reinforcement learning.

[12] Li, Y. (2017). Deep reinforcement learning: An overview.

[13] Lin, L.-J. (1992). *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Pittsburgh, PA, USA. UMI Order No. GAX93-22750.

[14] Lopes, M., Lang, T., Toussaint, M., and yves Oudeyer, P. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 206–214. Curran Associates, Inc.

[15] Machado, M. C. and Bowling, M. (2016). Learning purposeful behaviour in the absence of rewards.

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

[17] Mohamed, S. and Rezende, D. J. (2015). Variational information maximisation for intrinsically motivated reinforcement learning.

[18] Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning.

[19] Oudeyer, P.-Y. (2018). Computational theories of curiosity-driven learning.

[20] Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286.

[21] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. *ArXiv e-prints*.

[22] Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., and Young, S. (2007). 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*, NAACL-Short '07, pages 149–152, Stroudsburg, PA, USA. Association for Computational Linguistics.

[23] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay.

[24] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization.

[25] Sorg, J., Singh, S., and Lewis, R. L. (2012). Variance-based rewards for approximate bayesian reinforcement learning.

[26] Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models.

[27] Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

[28] Su, P.-H., Gasic, M., Mrksic, N., Rojas-Barahona, L., Ultes, S., Vandyke, D., Wen, T.-H., and Young, S. (2016). On-line active reward learning for policy optimisation in spoken dialogue systems.

[29] Sutton, R. and Barto, A. (1998). Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054.

[30] Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. (2016). #exploration: A study of count-based exploration for deep reinforcement learning.

[31] Ultes, S., Rojas Barahona, L. M., Su, P.-H., Vandyke, D., Kim, D., Casanueva, I. n., Budzianowski, P., Mrkšić, N., Wen, T.-H., Gasic, M., and Young, S. (2017). PyDial: A Multi-domain Statistical Dialogue System Toolkit. In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78, Vancouver, Canada. Association for Computational Linguistics.

[32] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay.

[33] Weisz, G., Budzianowski, P., Su, P.-H., and Gašić, M. (2018). Sample efficient deep reinforcement learning for dialogue systems with large action spaces.

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

[35] Young, S., Gasic, M., Thomson, B., and Williams, J. D. (2013). POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.

Appendix A

Action Space

Actions the PyDial dialogue system can take in the CR domain:

- **request + slot**, where slot is food, area or pricerange and the system prompts the user to specify their criteria for the specific slot requested, e.g. "What kind of food did you have in mind?"

- **confirm + slot**, where slot is an informable slot that can take any possible value for food, area or pricerange available in the ontology that may have been mentioned by the user. The system is using the confirm action in order to increase certainty in the users criteria, e.g. "Did you say you are looking for an Indian restaurant?"

- **select + slot**, where slot again is an informable slot that can take any possible value for food, area or pricerange available in the ontology. The select action gives the user a list of those values to select from, e.g. "Did you say you are looking for an Indian or Italian restaurant?" or "Would you like Japanese or Korean food?"

- **inform + method + slots**, where method specifies how the information about the restaurant is given, and informable slots for the restaurants are: area, food type, description, phone number, pricerange, address, postcode and signature. The user can request a restaurant based on all of these slots and request information for all of those slots, while the system will only request information about food, pricerange and area and inform on at least the restaurants name. Overall the system has 2^8 different ways to inform on a restaurant. The standard method is to chose the first restaurant in the ontology meeting all user criteria specified and is called *requested*. The two other methods available for the CR domain are *byname* and *alternatives*. Method *byname* is used when the system believes that the user asked about a specific restaurant by saying its name. If method is *alternatives* the system choses alternative restaurant meeting

the user criteria mentioned before. E.g. "Anatolia is a nice restaurant serving Turkish food, its telephone number is 01223 362372."

- **repeat** prompts the user to repeat its response, e.g. "Sorry, I couldn't understand that. What did you say?"
- **reqmore** prompts the user to provide more input, e.g. "Do you need any other information?"
- **bye** is the action to end the call usually used as response to the users intention to end the call.

Appendix B

Example Dialogue

For the system actions summary and master action are displayed
as summary_action → master_action.

Sys: hello() → hello()
User: inform(type=restaurant)
Sys: request_food() →request_food()
User: inform(food=indian)
Sys: inform() → inform(name="prana", area="center")
User: reqalts()
Sys: inform_alternatives() → inform(name="tiffin truck", food="indian")
User: request(phone)
Sys: inform_requested()→ inform(name="tiffin truck", phone="01223 366111")
User: thankyou()
Sys: bye() → bye()

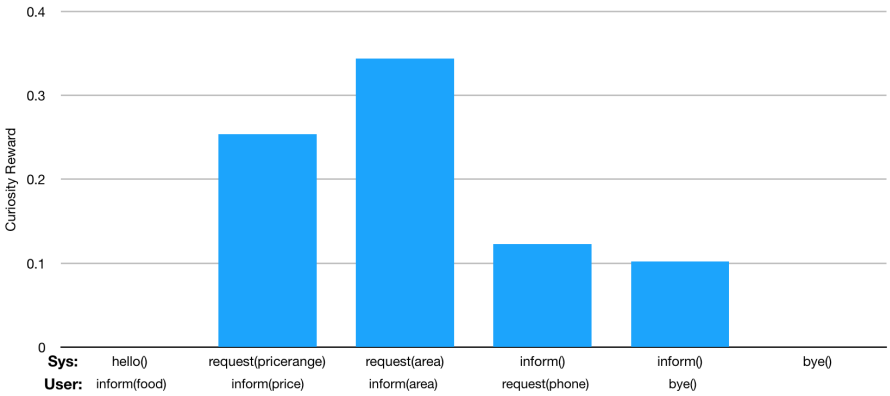


Figure B.1 Example dialogue with corresponding curiosity rewards.