



TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Intrinsic Motivation Complex for Chatbots

| | |
|--------------|---|
| Authors | Shreyash Agarwal, Emanuel Deisler, Oğuz Gültepe, Katharina Hermann, Lennart C. Neumann, Nina Schmid and Nicolas Seppich |
| Mentor | Olena Schüssler M.Sc. Steering Lab by Horváth & Partners GmbH |
| Co-Mentor | Cristina Cipriani M.Sc. (Department of Mathematics) |
| Project Lead | Dr. Ricardo Acevedo Cabra (Department of Mathematics) |
| Supervisor | Prof. Dr. Massimo Fornasier (Department of Mathematics) |

Feb 2021

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction: "Motivation" in Psychology and CS | 3 |
| 2 | Project Outline | 4 |
| 2.1 | Project Motivation | 4 |
| 2.2 | Project Goals and Requirements | 5 |
| 2.3 | Outline | 6 |
| 3 | Theoretical Background | 7 |
| 3.1 | The Reinforcement Learning Problem | 7 |
| 3.2 | Credit Assignment Problem | 8 |
| 3.3 | Exploration – Exploitation Dilemma | 8 |
| 3.4 | Q-Function and Bellman equation | 9 |
| 3.5 | Deep Reinforcement Learning | 10 |
| 4 | Baseline Agents | 12 |
| 4.1 | Intent Based Agent | 12 |
| 4.1.1 | User Simulator and Error Model Controller | 13 |
| 4.1.2 | DQN Agent | 15 |
| 4.1.3 | State Tracker | 16 |
| 4.1.4 | Training and Results | 16 |
| 4.2 | Multicomponent Agent | 16 |
| 4.2.1 | General Concept | 18 |
| 4.2.2 | Pipeline | 18 |
| 4.2.3 | Training and Results | 19 |
| 4.3 | End-to-end Agent | 20 |
| 4.3.1 | Data Generation | 20 |
| 4.3.2 | Agent Network | 21 |
| 4.3.3 | Training and Testing | 22 |
| 4.3.4 | Results | 22 |
| 4.4 | Selection of one Baseline Approach | 23 |
| 5 | Concepts for Intrinsic Motivation | 24 |
| 5.1 | Mood-model Based Intrinsic Motivation | 24 |
| 5.1.1 | Overall Concept | 24 |
| 5.1.2 | User Mood | 24 |
| 5.1.3 | Additional Extensions to the Baseline Intent Based Agent | 26 |
| 5.2 | Curiosity-driven Intrinsic Motivation | 26 |
| 5.2.1 | Motivation | 26 |
| 5.2.2 | Implementation | 27 |
| 6 | Experiments | 30 |
| 6.1 | Metrics | 30 |
| 6.1.1 | Task Success | 30 |
| 6.1.2 | Dialogue Efficiency | 31 |
| 6.1.3 | Reward Metric | 31 |

| | | |
|----------|--|-----------|
| 6.1.4 | Quality Metric | 32 |
| 6.2 | Hyperparameter Experiments | 32 |
| 6.2.1 | Intermediate Results | 33 |
| 6.3 | Final Results | 33 |
| 6.4 | Comparison Intrinsic Motivation Concepts | 34 |
| 6.4.1 | Comparison Metrics | 35 |
| 6.4.2 | Analysis of Conversation Examples | 36 |
| 6.5 | Continuous skill expansion | 36 |
| 7 | Conclusion and Future Work | 38 |
| A | Appendix | 40 |
| | References | 44 |

1 Introduction: "Motivation" in Psychology and CS

As humans we are not only capable of interacting with our environment, but we can also choose which of a nearly infinite pool of actions we want to perform. The question why we choose specific actions, instead of others can be answered with the concept of motivation. Motivation can be summarized as the active orientation of our current state of being towards a positively assessed goal state, or away from a negatively assessed event [1]. This means that there are two essential components to choosing a certain action: (1) a punishment and reward system, that gives feedback on whether the action consequence was positive or negative for us. This helps us build an experience base and make better decisions next time. And (2) the value that we attribute to a certain action consequence, combined with our expectancy that this consequence will actually occur [2], in order to maximise the probability that we will achieve the best possible outcome. These evaluations are dependent on the interaction of personal and situational factors, i.e. our personal motives and goals that we want to achieve with our actions, as well as the situation which provides certain opportunities and incentives [3].

This base model of motivation can be regarded as the direct inspiration of reinforcement learning (RL) in computer science. Reinforcement learning tries to allow a computer program to learn how to behave within a given environment. This learning process is based on the provision of positive or negative rewards for certain actions. The goal of the agent is to achieve the highest cumulative reward in the end [4]. Like in the aforementioned model of motivation, the actor, or agent, is fed situational information from the environment and tries to choose an action strategy (or "policy") that promises the greatest long-term reward, which ultimately implies goal completion (for a more detailed explanation see chapter 3.1).

However, there are some restrictions to this simplified model of motivation. As the reward ultimately comes from the completion of the goal, the actions are merely tools for the achievement of the pre-specified goal-state [1]. This means that learned strategies are inflexible and rather exclusive to this one scenario, as they are only known within this scenario.

Contrary to that there are actions that we perform daily, where the performance of the action itself is the goal, for example playing the piano. We trigger the action without being told by anybody to do so. This is an example of motivation existing along a spectrum from intrinsic (performing an action for the sake of itself) to extrinsic motivation (performing an action purely for the positive action outcome/the nonappearance of a negative action outcome) [3].

Furthermore, there is a capability that humans have, which is essential for making sense of the world around us. That is our intrinsic motivation to explore the world, even if there are no immediate practical benefits: curiosity. The basic neurological reward system, which is central for motivational processes as mentioned before, can be placed in very specific anatomical areas in the brain. Neuroimaging studies have found that the same areas that are part of this reward system are activated when we receive or anticipate information that we are curious about [5]. This means that information-seeking is neurophysiologically rewarded like any other motivated process [5].

To summarize, in the context of learning it is curiosity and intrinsic motivation that give us rewards for exploring our environment and our agency within this environment.

This concerns not only skills that we acquire [1] but also knowledge about interactions between the actor and the environment [3] and, to go one step further, dynamics within the environment. Humans do this constantly and autonomously, without relying on an explicit goal, leading to a life-long development and adaptation. Gaining such knowledge allows the flexible use of learned skills and information in different settings. It enables the actor to make experiences that can be useful to complete future goals in a more efficient and effective manner.

While the crucial impact of intrinsic motivation is well understood within the study of 'natural intelligence', in the context of artificial intelligence (AI), possible concepts and applications of intrinsic motivation are yet to be fully explored. However, there are known problems within RL research that stand out as potential candidates that could benefit from applications of intrinsic motivation (c.f. Eppe et al. 2020 [6]), one example being the sparse-reward-problem: As the complexity of the reinforcement learning environment grows, the event of "accidentally" achieving a goal state becomes increasingly unlikely. This in turn results in extremely scarce or even absent rewards, providing insufficient information to update the policy and severely dampening the efficiency of the agent [7]. Overcoming this problem requires an adaptation, rendering the agent capable of successful exploration [7], while still being able to fulfil the goal efficiently. As such, the sparse-reward-problem poses as a prime candidate to apply intrinsic motivation.

Getting a sense of achievement for gaining information about how to interact with the environment and affordances of objects, instead of the reward being provided extrinsically (which can be rare or even absent) is crucial for our ability to act.

But if curiosity and intrinsic motivation have such an existential meaning in natural intelligence, why wouldn't the correspondence in AI be of similar importance?

2 Project Outline

2.1 Project Motivation

Despite a great interest in moving digital agents from extrinsically motivated learning (as is the case in traditional reinforcement learning) to a more intrinsically motivated form due to the reasons previously discussed, concepts of curiosity and intrinsic motivation exist only sporadically and are still severely underrepresented to date [6].

With artificial intelligence having become integrated in all aspects of human existence this issue has a nearly unlimited scope in our society. In our daily life we most consciously interact with AI in the form of chatbots. One only has to think about the navigation systems in our cars or the automated voices in support hotlines to realize how naturally integrated they have become in our lives.

However, it is chatbots like Amazon's Alexa or Apple's Siri that make the major difference, as they not only work off one order, but support us in a variety of diverse every-day tasks across a long time. For this reason, it is especially crucial for them to be able to adapt to user-preferences and understand what the user needs throughout the day. An important point within this ability is to be able to tackle certain tasks that are necessary without being implicitly told to do so, as this would solve problems for the user before they even occur. Maybe you do not want to go to work by bike today because it's raining, and

your chatbot has already searched for the right transit connection and set your alarm accordingly? Maybe you always drink a cappuccino at 4 p.m. and the chatbot learns to order it for you, so that it always arrives on time?

These are abilities that are still lacking in modern chatbots and that could be augmented into a vast range of different directions. Following a bionic approach and emulating natural intelligence could lift AI onto a new level. Once such approaches could be developed and tested in one setting they could be adapted into any other setting AI has to offer.

2.2 Project Goals and Requirements

In order to see whether such approaches are feasible and useful, the aim of this project was to develop a chatbot that supports the user in fulfilling his task and carries on coherent conversations, and then augment it with an Intrinsic Motivation Complex (IMC). The IMC should fulfil the following requirements:

1. Ability to trigger actions itself
2. Continuous skill expansion

According to a certain school of psychology, motivation can be illustrated as a person moving within a force field, with positive and negative forces pushing or pulling this person towards a goal state, depending on the quantifiable need to achieve this goal state (c.f. [8]). As these forces can be expressed in mathematical formulae (depending on the intrinsic tension/need of the person to achieve and the distance from the goal state, c.f. Lewin's theory, summarized by Beckmann and Heckhausen [8]) an initial idea for the project contemplated to make use of this. However, this led to the question how the concept of needs could be translated into artificial actors. To answer this question we drew from Maslow's hierarchy of needs [9]. It posits that there are five sets of goals that are arranged in a hierarchy, whereas those lower in the hierarchy are regarded as more existential (e.g. food, safety) and prerequisites to achieve the higher goals (e.g. esteem, self-actualization). Once the goals of one hierarchical level are fulfilled, the action orientation of the individual switches to the next level of goals. An adaptation of such a concept for artificial intelligences could place basic requirements like battery charge on the lowest, user goal fulfilment at the medium and exploration and learning at the highest level of a needs hierarchy. This could enable the agent to learn to self-trigger and adapt actions in order to advance from one level to the next, as these needs are of intrinsic value to the agent without explicit solutions to the problems. It would also provide a task prioritization system that lies in the nature of the architecture. However, at this point two questions arose: is this a feasible and really efficient concept to implement in AI to fulfil our project goals? And is it ethical to develop AI agents capable of learning with needs. As we answered both questions with "no" we sought other options to achieve our project goals and returned to more basic motivation and machine learning concepts:

We came to the conclusion that the first requirement could be solved using the concept of intrinsic motivation generally: if something is "of intrinsic worth" to the agent (i.e. it provides a high reward within an RL framework) and there are not explicit solutions to the problem, this is reason enough for the agent to trigger a corresponding action, even if

it is not explicitly being told to do so. Well designed RL punishment and reward systems can teach the agent to do so only in appropriate situations.

The second requirement can be solved with curiosity specifically. Rewarding the agent not only for goal completion, but also for the exploration of the environment and its dynamics allows him to develop a more holistic experience base and therefore act more efficiently in the future. However, this cannot get into the way of the completion of the task at hand. According to the reward system's dynamics in the human brain both requirements should be able to be fulfilled using similar reward dynamics in reinforcement learning agents.

2.3 Outline

In order to develop a prototype IMC, we chose an example use case for an agent with such an augmentation that was close to an everyday user and (for now) limited to one setting. This example use case is that of a bar tender chatbot. In the future this project can be used as the basis to extend the IMC to more complex and diverse use cases, arguably having a chance to contribute to change in all RL agents.

In the following report, we explain the theoretical details of reinforcement learning and deep learning in more detail. Subsequently we describe our implementations of baseline agents which we developed to be extended with an IMC. Then we introduce the concepts and implementations of our solutions for intrinsic motivation, which subdivide in a mood-model, which is used for the self-trigger-requirement of the agent and a curiosity-model, which is used for the continuous-learning-requirement. Furthermore, we report on findings gained from experiments we conducted to prove the efficacy of our implementations, conclude on the feasibility of an IMC as we tried to build it and give an outlook to future research based on our work.

3 Theoretical Background

The basic concepts of reinforcement learning as a separate branch of machine learning is briefly described in the following pages.

3.1 The Reinforcement Learning Problem

Reinforcement learning (RL) is an approach to solving the task of learning. This is to be achieved through interaction with the environment with the intent to achieve a goal. The learner and decision maker is called the agent. This agent interacts with an environment through a clearly defined interface. The agent learns from the feedback of its actions that the environment presents [10]. The agent may have no knowledge of the inner workings of the environment. The naming convention is similar to control theory where a controller (agent) makes changes to a control system or plant (environment) via control signals (actions) [11].

The agent and the environment interact continually at discrete time steps t . At each time step the agent gets as an input the state of the system (consisting of the environment and the agent) $s_t \in S$, where S is a set of existing states. Based on the state s_t the agent selects an action $a_t \in A(s_t)$. $A(s_t)$ being a set of legal actions in that state. Based on this action a_t the agent interacts with the environment, transitioning the the system into a new state $s_{(t+1)}$, which is then fed again as an input to the agent. The transition from one state into another is accompanied by a reward $r_{(t+1)} \in R$, which is a usually a scalar value. The number of time steps can be finite or infinite. An infinite number of time steps form a continuous learning task. A finite number of time steps make up a learning or training episode, which is what has been chosen for this project. The overall objective is to learn a policy $\pi(s_t)$ to choose actions, which maximise the reward over time (or over the duration of a training episode) [11]. $\pi(s_t, a_t)$ is the probability of choosing action a_t in the state s_t .

The aim of reinforcement learning is to improve the policy of the agent as a result of the feedback the agent receives from the environment in form of the reward r_t [11].

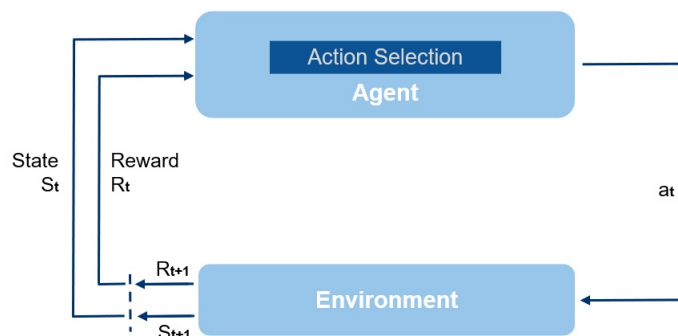


Figure 1: Agent environment interaction

In order to determine whether an action was directed towards the aforementioned goal, the reward $r_t \in R$ is given encoding the success. The goal is to maximise the reward for the entire episode, this means that not the immediate reward is relevant but the

accumulated reward over time. Using a reward to formalise the objective of the task is one of the distinctive traits of reinforcement learning [12].

Ideally no knowledge on how a problem it to be solved should be imparted into the reward signal but only indications of what should be accomplished so the search space for an ideal solution is not limited and to the agent is not constrained. Reinforcement learning environments are usually formalised as a markov decision process (MDP) [13].

3.2 Credit Assignment Problem

The credit assignment problem describes the fact that rewards for actions are usually extremely delayed. An agent has to learn not to take the action that gives the immediate highest reward but go through a number of states with almost no or zero reward in order to get to more relevant and thus more rewarding states in the future. As a consequence the reward signal is weakened over the number of steps taken [10]. For a finite number of steps T a common way to overcome this problem is to use a discounted reward function:

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

t being the steps and k the number of times a certain reward is obtained. This states, that the strength by which an action is encouraged or discouraged is the weighted sum of all rewards of one episode. R_t is the total discounted reward that can be expected if a certain action $a \in A(s_t)$ is chosen. r_t is the reward for the action at time step t . γ is a parameter between $0 < \gamma < 1$ called the discount rate. Hereby, the later rewards are exponentially less important. The larger γ is chosen, the further into the future are rewards taken into account [14, 15].

3.3 Exploration – Exploitation Dilemma

In order to maximise the reward over time, the agent has to explore the environment. This means the agent has to explore as many states as possible in order to “know”, how being in a specific state will influence the overall reward. In an environment with very large or even an infinite set of possible states, exploring every state can take a very long time. Therefore, a rule has to be formalised how the agent will transfer from exploring the environment to exploiting the policy it has found. If a reinforcement algorithm is initialised randomly, the actions chosen in the beginning will be random. Here, the agent performs a sort of “built-in” exploration. However, once the agent has found a policy giving it a maximum reward, it will continually use this policy, shifting to exploiting the policy. The maximum reward found, may however not be the maximum achievable reward in the environment. This exploration is called greedy and settles on the first policy found [11].

There are different ways to formalise how the agent should shift from exploration to exploitation. One of them is a ϵ – greedy exploration. Here the agent will pick the best action according to its policy but with a $(1 - \epsilon)$ chance will pick a random action. Usually ϵ is annealed over time to privilege exploitation over exploration, after sufficient exploration of the environment [13].

3.4 Q-Function and Bellman equation

A common method for solving reinforcement learning tasks is based on what is called the Q-function. At each time step the agent interacts with an environment, it chooses an action which moves it from its current state into a new state. Additionally, the agent receives a reward. The agent tries to choose the actions that will yield it the maximum reward for an episode. In order to choose the action in each state that yields the maximum expected reward at the end of an episode an optimal value action function $Q^*(s, a)$ is introduced. The Q-value function at state s_t and action a_t (see eq. 2 and eq. 3) is the expected cumulative reward from taking action a_t in state s_t and then following the policy π . The symbol Q stands for quality since it represents how advantageous a certain action is in a certain state [16].

$$Q(s, a) = \max_{\pi} E [R_t | s_t = s, a_t = a, \pi] \quad (2)$$

With eq. 1 this results in:

$$Q(s, a) = \max_{\pi} E \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi \right] \quad (3)$$

The policy π for choosing an optimal action to a state, is chosen as the following:

$$\pi(s) = \operatorname{argmax}_a Q(s, a) \quad (4)$$

Since the Q-function is usually randomly initialised (Q^* is unknown at the beginning), the optimal action values for the Q-function have to be found. This is possible because the defined optimal action value function obeys an identity known as the Bellman equation [16, 10]:

$$Q^*(s_t, a_t) = E \left[r_{t+1} + \gamma \max_a Q^*(s_{t+1}, a_{t+1}) \right] \quad (5)$$

The basic idea is that the maximum possible reward (this is under optimal policy) for the current state and action is obtained, plus the maximum discounted future reward of the next state. Essentially the bellman equation establishes a link between present and future states. In order to find the optimal actions in each state, the Q-function is iteratively approximated using the Bellman equation. This is often referred to as Q-learning [11, 10, 17]:

$$Q_{i+1}(s_t, a_t) = Q_i(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a_{t+1}} Q_i(s_{t+1}, a_{t+1}) - Q_i(s_t, a_t) \right], \quad (6)$$

where i is the update iteration and α a factor known as the step size. It has been shown that an algorithm performing such value iteration will converge to an optimal policy [18].

$$Q_i \rightarrow Q^* \text{ as } i \rightarrow \infty \quad (7)$$

Updating $Q(s, a)$ using $\max_a Q[s_{t+1}, a]$ is only an estimation at first, but performing such an update often enough will lead to the convergence of the Q-Function to represent the true Q-Value [16].

3.5 Deep Reinforcement Learning

The theoretical background of artificial neural networks is assumed to be known and will thus not be explained in detail in this report. The following chapter describes how NNs are integrated into the reinforcement framework. The Q-learning approach has the objective of determining optimal action values to maximise the overall reward [11, 12].

In practice the reinforcement learning approach described in Chapter 3.4 is impractical [16] because the Q-values are evaluated for each state action pair, which is infeasible for a very large state-action space [15].

Instead, a function approximator is used to evaluate the Q-values. This can be a linear or a nonlinear function approximator such as a NN.

The implementation of a NN as a function approximator for the Q-learning approach was pioneered and implemented by DeepMind [19]. The Q-network is, like a standard NN, trained by minimising a loss L through an iterative process based on Q-learning with the underlying Bellman equation:

$$L_n = E \left[\left(r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{n-1}) - Q(s_t, a_t; \theta_n) \right)^2 \right] \quad (8)$$

$Q(s_t, a_t)$ is the prediction calculated from the input state s_t , which is a multihead output for each action. The actual action a_t is then received as the optimal policy by:

$$a_t = \pi(s_t) = \underset{a_t}{\operatorname{argmax}} Q^*(s_t, a_t; \theta_n) \quad (9)$$

In equation 8 $r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}; \theta_{n-1})$ is the target value y_i - the ground truth in supervised learning. Minimizing this loss means that the Q-value function learned by the network iteratively approaches the target y_i , which is the calculated optimal Q*-value function approximating the Bellman equation 5.

This target value is obtained by executing the actions a_t chosen from $Q(s_t, a_t; \theta_n)$ in the environment, getting state s_{t+1} and the reward r_{t+1} and running s_{t+1} through the so called target Q-network getting $Q^*(s_{t+1}, a_{t+1}; \theta_{n-1})$. In contrast to the label, which was seen as the unchangeable ground truth in supervised learning, the target Q-network is dependent on the weights of the Q-network [16, 15], but its weights θ_{n-1} are always lacking one step behind the weights θ_n of the actual Q-network in order to increase stability.

Differentiating the loss function 8 with respect to the weights θ of the Q-network produces a gradient and leads to the following expression [11, 16]:

$$\nabla_{\theta} L_i = E \left[\left(r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{n-1}) - Q(s_t, a_t; \theta_n) \right) \nabla_{\theta_n} Q(s_t, a_t; \theta_n) \right] \quad (10)$$

The deep Q-learning approach is an off policy learning approach. The network does not learn a policy. The network only learns the Q-value for each action, state pair as described in previous chapters 3.3 and [16, 15]. The term off-policy means that the Q-function is updated independently from the current episode with samples, which have been saved

in a replay buffer throughout the exploration and are now sampled with the assumption that they are independently identically distributed (iid). These samples include the state s_t , the action a_t , the next state s_{t+1} and the reward r_t .

To calculate the Q-values of the actions the agent has at its disposal, a naïve form of constructing the model would be to use the state and an action as input to the neural network to produce a Q-value for that state and that action. However, if the agent has several actions to choose from, a separate forward pass of the network is required for each Q-value. This produces a computing cost that scales linearly with the number of actions. To avoid this, the architecture is changed to have one output per possible action. Each output represents the Q-value of one action with the state as the only input to the network.

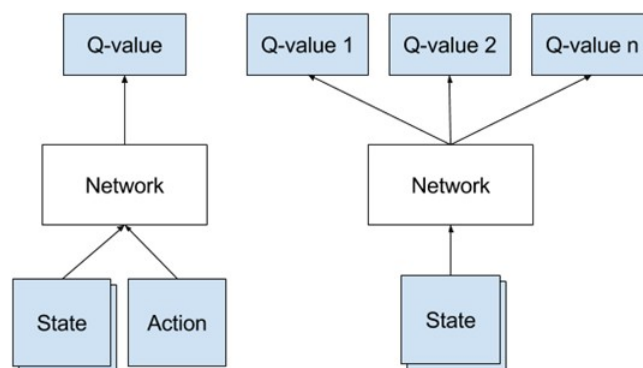


Figure 2: Left: classic Q-learning with one Q-values per calculation; right: Q-network with one Q-value per output

4 Baseline Agents

Our chosen use case for this work is a goal-driven bartender agent. The task each approach has to fulfill for the baseline is to serve the user a drink throughout the conversation, which matches constraints such as drink type or size, given by the user. We have three approaches in total, the first one is based purely on intents, while the second and the third operate on a natural language level.

4.1 Intent Based Agent

The intent based agent is a value-based reinforcement learning agent. The problem statement is the following: At the beginning of a conversation - a so called episode - the user picks a goal from a goal data base of the following form : $\{ 'request_slots' : \{ 'SIZE' : 'UNK' \}, 'inform_slots' : \{ 'TEMP' : 'cold', 'DRINK' : 'water' \} \}$.

The goal has so called request and inform slots filled with attribute slots, which themselves are filled with values. The request slots indicate which information has to be informed by the agent throughout the conversation and the inform slots indicate which constraints the user poses for his drink. Throughout one episode the agent and the user then interact in steps via so called utterances of the form shown in figure 3.

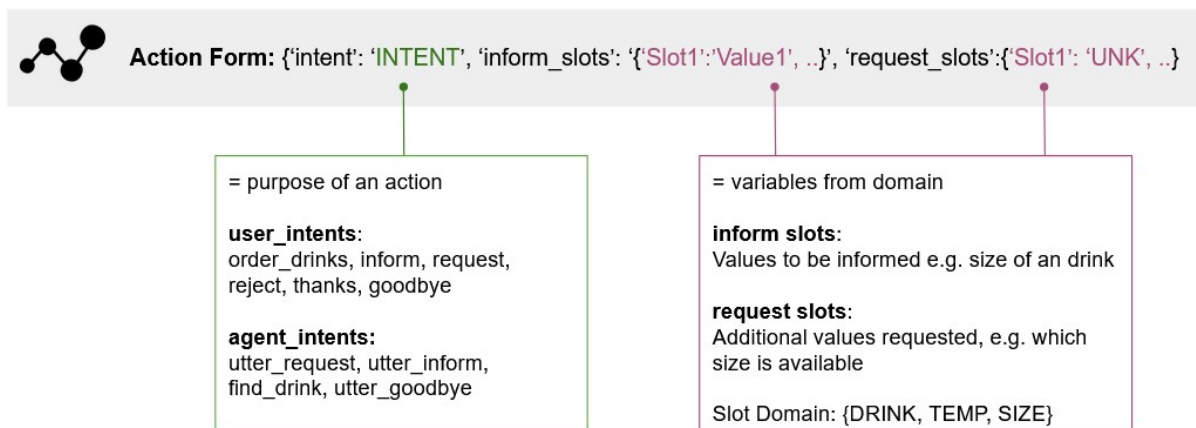


Figure 3: General action utterance of either the agent or the user

The intent indicates the purpose of the conversation, while the request and the inform slots again indicate which information is requested or informed. As figure 3 shows, the user has 6 intents in total, and the agent has 4. The slots domain has a size of 3 with the drink type, the temperature of the drink and its size. Each slot has a value set (e.g. $\{ small, medium, large, Jumbo \}$ for the *SIZE* slot), which is filled in case the slot is part of the request slots.

The overall goal of the conversation (one episode) for the agent is to inform all the unknown values from the goal's request slots and to find out all the constraints at the same time, to search for a drink in the database matching all the slots values. Once the agent has found a drink, he will inform it to the user, who indicates whether the search has been a success or not. Ideally the agent then closes the conversation, which is the end of an episode and gets a high positive reward. If the agent doesn't close the conversation, it

continues until a maximum step number of an episode (turns in a conversation) is reached and the user simulator closes the conversation automatically.

The overall agent’s experience collection process for the DQN training is conducted in a so called gym, which can be seen in figure 4. The architecture of this process follows the approach from [20].

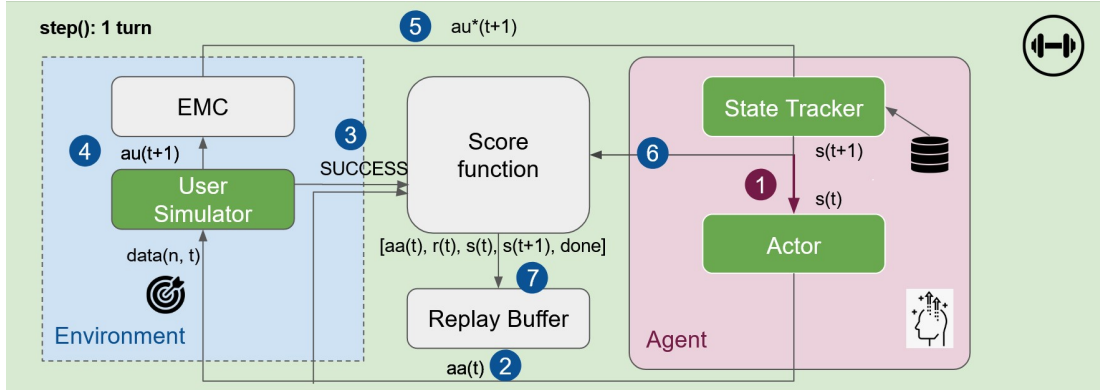


Figure 4: Gym: Place, where the agent can explore its state action space and learns its policy

As a first step the agent starts with a state $s(t)$, that encodes information about the history of the conversation. Based on this state, the DQN agent (see 4.1.2) chooses an action $aa(t)$ (corresponding to an utterance) the agent wants to take based on its current policy. This agent action $aa(t)$ is then forwarded to the user simulator (see 4.1.1), which in the reinforcement context is the environment the agent interacts with. The user simulator reacts with a rule based user utterance $au(t)$ to the agent action and also outputs a SUCCESS variable indicating, whether a drink has already been found. The SUCCESS variable is then forwarded to the score function, where the reward for the agent is computed.

The user utterance $au(t)$ on the other hand is forwarded to an EMC (error model controller) infusing an error to the user utterance with a certain probability, improving the training stability. This action $au^*(t)$ is then forwarded to the state tracker (see 4.1.3). The state tracker first updates its own history of the conversation, which it keeps track of and then uses this history information to compute the state $s(t + 1)$ of the agent.

This state $s(t + 1)$ is then together with the state $s(t)$, the agent action $aa(t)$, the reward $r(t)$, and the *done* variable (indicating whether one conversation round (episode) is finished) forwarded to the replay buffer, which is the memory of the agent, from which the DQN agent learns its policy. The state $s(t + 1)$ is again forwarded to the DQN agent, which closes one experience loop or step in an episode.

In the following the most important components, the user simulator, the DQN agent, the state tracker and the error model controller (EMC) will be explained in more detail.

4.1.1 User Simulator and Error Model Controller

The user simulator (see 4.1.1) is the environment the agent interacts with at each step in order to get a reward for certain actions.

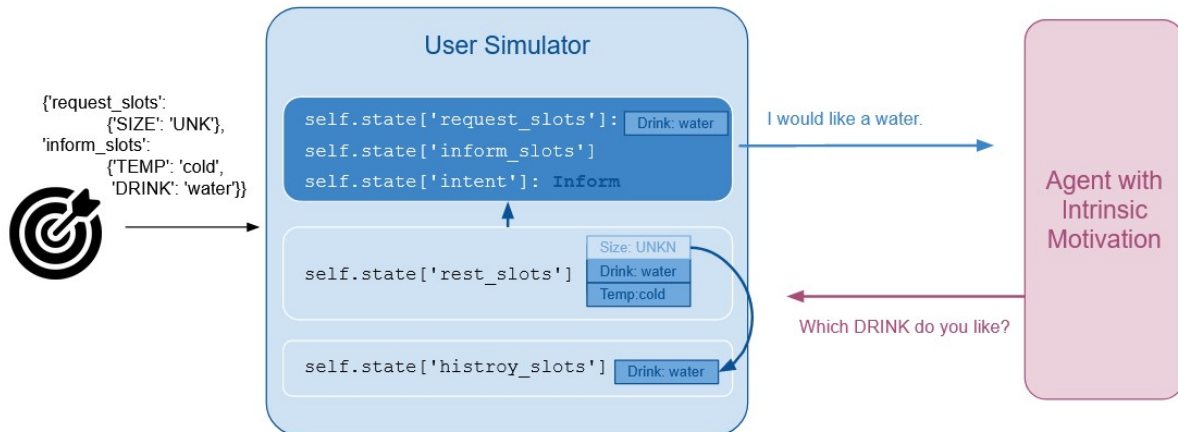


Figure 5: User simulator with its different states keeping track of the conversation: example of a user inform utterance to a drink request agent utterance

The user simulator acts based on internal states, which represent the user’s memory. As explained in 4.1 at the beginning of a conversation the user chooses a random goal from a goal database. The request and inform slots of this goal are then saved in the rest slots state, indicating which slots still have to be informed or requested by the user throughout a conversation. The history slots state indicates, which slots have already been informed throughout an episode by either the user or the agent. Whenever the user simulator receives an utterance from the agent $aa(t)$ with certain intent and inform and request slots, it chooses a rule based answer to this certain action. This answer depends on the agent action’s intent, its request and inform slots. If the agent as in 4.1.1 for example requests for a DRINK slot, which is still in the rest slots state (meaning it hasn’t been informed throughout the conversation yet), the user picks this slot. Since the slot value is filled with an actual value and not with an unknown, the action will have an inform intent. Once the user has chosen an intent and a slot from the rest slots, this information is filled in the request slots, inform slots and intent states forming the utterance of a user. This utterance is then passed on to the agent. As the DRINK slot will be informed by the user, it gets passed to the history slots state.

Once all rest slots are empty the user requests the drink itself. The respective slots of the drink, being proposed from the agent have then to match all the history slots to fulfill the goal.

Whether a matching drink is found is checked by the user only at the end of an episode, expressed through a positive or a negative SUCCESS value, which is then fed into the score function, computing the reward of the agent as in eq. 11. Nevertheless there is one exception for the case that the user closes the conversation itself, which happens if a maximum number of steps is reached. In this case the SUCCESS value will always be set to negative - independently of whether a drink has been found. This enforces the agent to close the conversation itself as early as possible. Throughout the rest of the conversation the SUCCESS value fed into the value function is always zero, which leads only to a fixed

negative reward of 1 for each additional step during the conversation, penalizing long and inefficient conversations (see eq. 11).

$$r_{extrinsic} = \begin{cases} -w_{fail} \cdot max_round & \text{if } SUCCESS = -1 \\ +w_{success} \cdot max_round & \text{if } SUCCESS = +1 \\ r_{step} & \text{if } SUCCESS = 0 \end{cases} \quad (11)$$

For completeness, the parameters w_{fail} , $w_{success}$ and r_{step} shall be understood as hyperparameter, since they will be considered as hyperparameter in subsequent chapters. However, at the baseline agent we kept them static at: $w_{fail} = 1$, $w_{success} = 2$ and $r_{step} = -1$.

The error model controller (EMC), introduces a certain randomness to the user answers by switching either the slot value, the slot itself or the overall intent with a certain probability, that can be chosen individually as a hyperparameter (see table 2).

4.1.2 DQN Agent

The DQN agent is the core of the RL chatbot. It is the part, which we try to optimize with the training process in our gym environment. The agent has a warm-up and an actual training phase. In the warm-up phase, the agent chooses its actions in a rule based fashion, with which it already collects experiences, which it can save in the replay buffer to initialize the training. As the training starts, the agent, which is in fact a neural network, gets as an input the state s_t as elaborated in 4.1. This network computes as an output the Q-value function of that state and each possible action $Q(s(t), aa(t)|\theta_n)$ as shown in figure 6 and mentioned in 3.5. The actual agent action $aa(t)$ is then chosen among all possible actions as the action, which maximizes the Q-value (see eq.9).

Improving the policy of the agent network for choosing an appropriate action $aa(t)$ to a certain input state in our case means that the neural network learns the optimal Q-value function $Q(s(t), aa|\theta_n)$. To find the optimum we want to minimize the Loss function 8, described in chapter 3.5), which for our concrete problem is the following:

$$L_n(\theta) = \frac{1}{2} \sum_{i=1}^{batch_size} (y_i - Q(s_i(t), aa_i(t)|\theta_n))^2 \quad (12)$$

with

$$y_i = Q^*(s_i(t), aa_i(t)|\theta_n) = \sum_{i=1}^{batch_size} (r_i + \gamma \max_{aa(t+1)} Q(s_i(t+1), aa(t+1)|\theta_{n-1})) \quad (13)$$

This loss is an approximation of the Q-Iteration as described in the theoretical background section of chapter 3.5. Having a deeper look at the DQN agent, we see that the network itself is rather simple, which makes the agent very efficient to train. The details about the architecture can be seen in table 4.

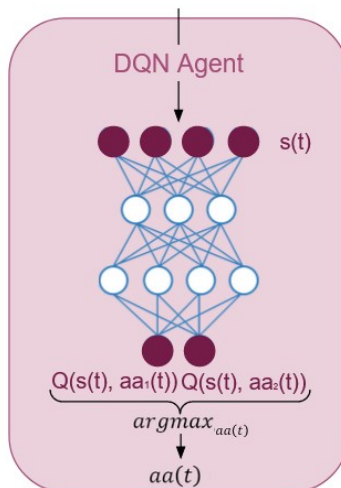


Figure 6: The DQN agent with its action selection Q-value neural network

4.1.3 State Tracker

The state tracker is tightly coupled with the DQN agent and part of the chatbot itself. The state tracker gets as an input the user action $au(t + 1)$. As shown in 7, with this action it first updates the dialogue history, which stores the last agent and user actions as well as all previous inform slots with its values. Secondly the state tracker computes the current state $s(t)$ based on the last user and agent action in the dialogue history.

This state is a one hot encoded vector, encoding and concatenating the following information: intent of last utterance, last slot informed, last slot requested by both user and agent, already filled slots, the current round number and a representation of database query results matching the current filled slot values.

This state $s(t)$ is then fed into the DQN, from which the state tracker gets back the agent action output $aa(t)$. The reason why the output is fed back to the state tracker is, that the agent chooses the action with its intent and its inform or request slot, but in case of an *inform* or *find_drink* action, it still has to fill the values for the slots by querying the database. This happens in the third step of the state tracker, where the agent searches for all database entries matching the current constraints and either return it for *find_drink* or for *inform* fill in the slot value, for which there are still the most entries. The state tracker then returns the filled agent action $aa(t)$ and updates the dialogue history with it before it passes it on to the user again.

4.1.4 Training and Results

The training took about 1-2h using a normal CPU, realized more or less coherent conversations and yielded a maximal success rate for finding a drink of 98%.

4.2 Multicomponent Agent

The intent based baseline model has an explicit mapping of utterances to the intents. Hence, it lacks natural language understanding (NLU) capability. Next to the intent based agent, two baseline agents that allow for natural language understanding were

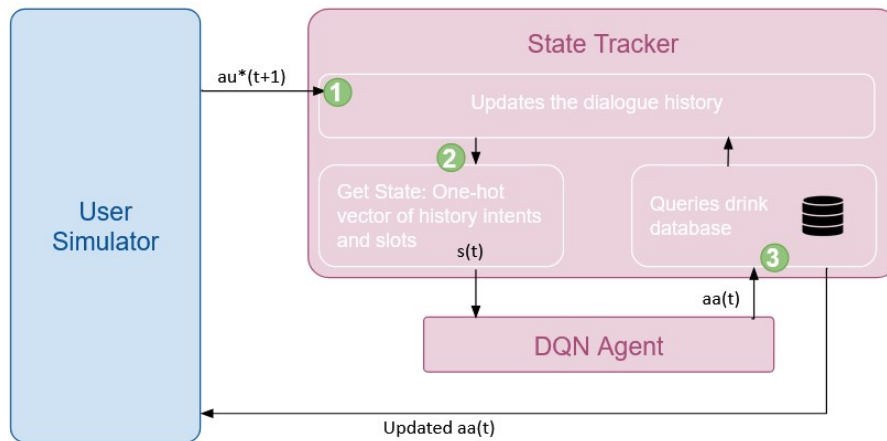


Figure 7: The state tracker with its three components to pre and post-process user and agent actions respectively to supplement the DQN agent

developed. One of them, the multicomponent agent, follows a reinforcement learning approach, and the other one, the end-to-end agent, is in the first stage being trained with supervised learning. The pipeline for both the models is inspired from [21]. Zhao et al [21] proposed a goal-oriented chatbot that can be trained end to end and combines the NLU, state tracker and dialog policy into one component. This helps in alleviating mainly two problems existing in the architectures which have each component separately.

- **Credit Assignment problem:** It's often difficult to identify the faulty component in the pipeline which is affecting the performance.
- **Process Interdependence:** Whenever one module (e.g. NLU) is retrained with new data, all the others (e.g. state tracker) that depend on it become sub-optimal since they were trained on the output distributions of the older version of the module.

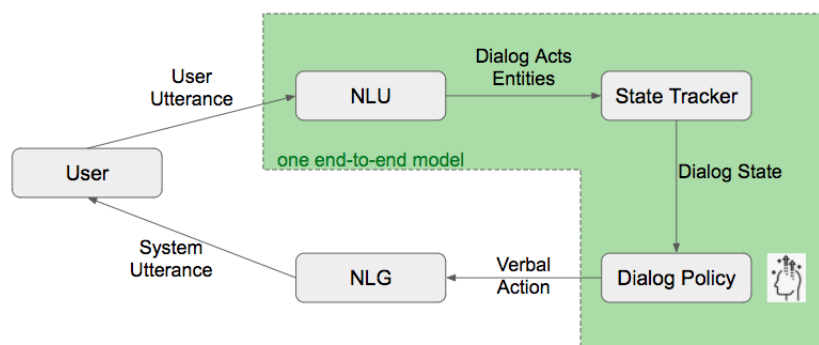


Figure 8: Conventional pipeline of a spoken dialog system (SDS). The proposed model replaces the modules in the dotted-line box with one end-to-end model.

This section presents the concept and results of the multicomponent agent.

4.2.1 General Concept

The general idea is an agent that interacts with the environment using two types of actions: verbal actions and hypothesis actions. Verbal actions are agent utterances, hypothesis actions represent the filling of a slot with a value. The environment consists of a user and a database (DB). Based on the agent action the user can return both an utterance and a reward to the agent. This concept (displayed in figure 9) is an adaption of reinforcement learning.

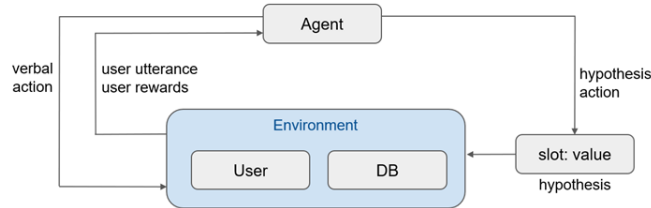


Figure 9: Multicomponent Agent: General Concept

The agent’s task is to have a coherent conversation and to fill the two existing slots *drink* and *size* correctly. There exist four types of user utterances: *greet*, *thanks*, *inform{\size}* and *inform{\drink}* producing utterances to greet, thank, inform the agent of the user goal’s size slot and inform the agent of the user goal’s drink slot respectively. Example utterances for each of the categories are: ‘hey there’, ‘I appreciate it’, ‘make it medium’ and ‘I would like to have a coke’. The agent’s verbal action can belong to one of four types: *utter_greet*, *utter_ask_DRINK*, *utter_ask_SIZE* and *utter_confirm_order*. Those verbal actions allow the agent to either greet, ask the user for the drink type, ask the user for the drink size or to confirm the order while stating the agent’s current believe about what kind of drink in which size to order. ‘Hey! Can I do anything for you?’, ‘What drink do you want?’, ‘What size do you prefer?’ and ‘Good choice. I will bring you a large frappe in just a second.’ are example agent utterances for each of those categories. An overview of those three pipeline components can be found in figure 10.

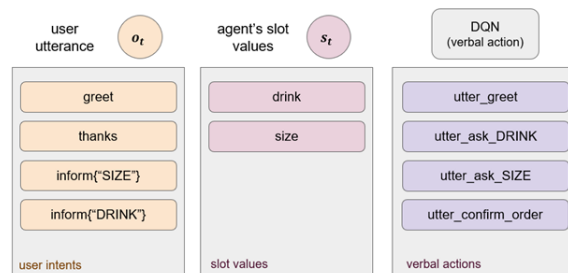


Figure 10: Overview of Pipeline Components

4.2.2 Pipeline

The multicomponent agent consists of one LSTM and three deep Q-Networks. The LSTM functions as state tracker and each of the DQNs corresponds to either a verbal action or

to one of the two hypothesis actions, i.e. the agent action to fill the drink slot or the size slot with a value. The LSTM gets two inputs: The agent’s slot values for drink and size whose tuple has the initial value of (0,0) and the user utterance combined with the user intent. During each of the agent actions only one of the three DQNs is being activated and the rest of them are masked. The action masking is based on the user intent and encoded as a deterministic mapping. To train the model efficiently a user simulation has been created. The training data consists of a set of stories, drinks and sizes. At the beginning of a new training conversation a story and a user goal consisting of a specific drink and size are randomly chosen. Thereafter the conversation flow is determined by the story. The general model structure is depicted in figure 11.

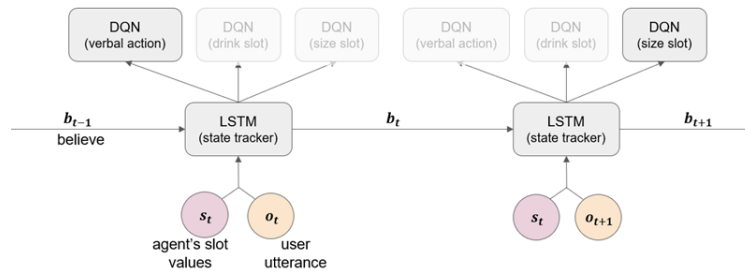


Figure 11: Pipeline of the Multicomponent Agent

When the story starts with a user utterance of the category *inform_drink*, e.g. ‘I want to order a frappe’ then the agent’s initial slot values (0,0) as well as this utterance are fed into the LSTM. Due to the user intent *inform_drink* the drink slot DQN is activated and the others are masked. With an optimally behaving agent the word ‘frappe’ is the output of this DQN and the agent’s slot value is updated to (‘frappe’, 0). Afterwards this tuple together with the previous user action is fed into the LSTM and the verbal action DQN is the one being unmasked. The network returns the agent action *utter_ask_size* and the agent therefore asks the user for the drink size. As a next step of the story the user’s intent is set to *inform_size* and the current agent’s slot values as well as a user utterance like ‘medium, please’ are passed to the LSTM. The size slot DQN is unmasked, the value ‘medium’ is added to the size slot and after another conversation step the verbal action *utter_confirm_order* is activated through the now unmasked verbal action DQN. The conversation episode ends and if both agent’s slot values correspond to the slot values of the user goal the agent gets a positive reward and otherwise a negative reward. If in any step of the conversation the agent fills a slot wrongly then the user simulation ensures that the conversation terminates and returns a negative reward.

4.2.3 Training and Results

The training takes about one to two hours using a regular CPU. Independent of training time, small implementation changes like a simplified database containing only a single story and fewer drinks and utterances the agent could not get over a success rate of over 15%. In figure 12 one can see two typical episodes containing agent errors. In the first episode the agent does not infer the information about the drink size correctly and in the second episode the agent does not choose the right verbal action, which, in this case, would be *utter_confirm_order*.

```

***** Episode 1676 *****
User Goal: 191 , 193
-----
Initial User Utterance: i would like to have fruit cooler
Agent Action: fruit cooler
Agent Action: What size do you prefer?
User Response: oh actually make it medium
Agent Action: large

***** Episode 1677 *****
User Goal: 186 , 194
-----
Initial User Utterance: i would like to have frapp
Agent Action: frapp
Agent Action: What size do you prefer?
User Response: oh actually make it large
Agent Action: large
Agent Action: What can I get you? :)
User Response:

```

Figure 12: Episode Examples for a Conversation of the Multicomponent Agent

Since all attempts to improve the agent performance failed to produce significantly better results, a second concept for a baseline agent allowing for natural language understanding was conducted.

4.3 End-to-end Agent

For our third baseline implementation, we train an end-to-end natural language understanding conversational agent in a supervised manner. We define our state to be consisting of the conversational state and the slots' state, denoting the observed dialog and the already filled slots, respectively. We furthermore define agent actions in three subsets: utter actions, drink slot actions and size slot actions, consisting of 5, 50 and 3 elements, respectively. The utter actions are the actions related to the conversation, consisting of *utter_greet*, *utter_ask_DRINK*, *utter_ask_SIZE*, *utter_confirm_order* and *utter_pass_turn*. The slot actions consist of each possible drink and size. The agent selects an action at every action turn, which is decoupled from the conversational turn. This allows the agent to take multiple actions on a single conversational turn. When the agent selects the action *utter_pass_turn*, the conversational turn is passed to the user.

4.3.1 Data Generation

In order to train our agent in a supervised manner, we first generate data simulating real conversations. This is done in two steps: Forming templates and generating data points. In the template forming stage, we go over the possible conversational stories provided to us by our mentor at Horváth (see fig.13) and form empty templates that correspond to the stories. A template consists of the conversational state, slot's state and the correct agent action at every point in the story. We form six templates in this stage, which are then used for template filling.

During the template filling stage, we sample from a total of 50 drink types and 3 drink sizes to fill the templates. We also sample for each utterance a natural language phrase from a pool of corresponding phrases. Filling the templates for every possible combination of drinks, sizes and utterances results in $\sim 29,000,000$ data points. We shuffle these data points and split them into a training set and a test set, where the training set includes $\sim 90\%$ of the data and the test set includes $\sim 10\%$.

```

#
* greet
  - utter_greet
  - pass_turn
* order_drinks
  - utter_ask_DRINK
  - pass_turn
* inform{"DRINK"}
  - slot{"DRINK"}
  - utter_ask_SIZE
  - pass_turn
* inform{"SIZE"}
  - slot{"SIZE"}
  - utter_confirm_order

```

Figure 13: A sample story

In order to facilitate data preprocessing and efficient data loading, we extend the **Keras Sequence** class into our own **ConvoSequence** class. This class converts conversational input into a **Numpy** array, and the slots' state input into a concatenation of 2 one hot encoded vectors of size 50 and 3. These vectors denote whether one of the 50 drink types or one of the 3 drink sizes are selected. The action output is also converted into a one hot vector of 58 dimensions, denoting the 5 utterance actions, 50 drink slot filling actions and 3 size slot filling actions.

4.3.2 Agent Network

The agent network consists of two inputs and three outputs. These correspond to the conversational input, slots input, and utter actions, drink actions and size actions, respectively. The inputs are processed to create meaningful input representations, which are then concatenated and passed to the policy networks to produce the outputs. The outputs from the policy networks are further passed through a softmax layer to form a probability distribution. The model summary can be seen in Fig. 14.

Conversational input Conversation input is initially passed through a vectorization layer which removes accents and punctuation, and converts the text to a sequence of integer indices. The resulting indices are passed through an embedding layer that projects the words onto a 300 dimensional embedding space. We initialise the embedding layer using pre-trained glove embeddings and then fine tune on our own data. Finally, the embedding representations are passed through an LSTM layer with a hidden state size of 300 to form our conversation representations.

Slots Input Slots states are provided as one hot encoded vectors of 53 dimensions. This input is then passed through a 100 dimensional dense layer with ReLU activation to form the slots representation.

Concatenation Layer The conversation and slots representations are concatenated into a 400 dimensional vector that is fed into the policy networks.

Policy Networks The policy network passes the concatenated state representation through two dense layers. For each network, the initial layer has a size of 200. The second layer has a size corresponding to the action space: 5, 50 and 3 for conversational policy, drink policy and size policy, respectively. All layers have ReLU activation.

Softmax Output The output of the three policy networks are passed through a softmax layer to form an action probability distribution.

| Layer (type) | Output Shape | Param # | Connected to |
|---------------------------------|-------------------|---------|---|
| input_1 (InputLayer) | [(None, None)] | 0 | |
| text_vectorization (TextVectori | (None, None) | 0 | input_1[0][0] |
| embedding (Embedding) | (None, None, 300) | 126300 | text_vectorization[0][0] |
| input_2 (InputLayer) | [(None, 53)] | 0 | |
| lstm (LSTM) | (None, 300) | 721200 | embedding[0][0] |
| dense (Dense) | (None, 100) | 5400 | input_2[0][0] |
| concatenate (Concatenate) | (None, 400) | 0 | lstm[0][0] dense[0][0] |
| dense_1 (Dense) | (None, 200) | 80200 | concatenate[0][0] |
| dense_3 (Dense) | (None, 200) | 80200 | concatenate[0][0] |
| dense_5 (Dense) | (None, 200) | 80200 | concatenate[0][0] |
| dense_2 (Dense) | (None, 5) | 1005 | dense_1[0][0] |
| dense_4 (Dense) | (None, 50) | 10050 | dense_3[0][0] |
| dense_6 (Dense) | (None, 3) | 603 | dense_5[0][0] |
| concatenate_1 (Concatenate) | (None, 58) | 0 | dense_2[0][0] dense_4[0][0] dense_6[0][0] |
| softmax (Softmax) | (None, 58) | 0 | concatenate_1[0][0] |
| Total params: 1,105,158 | | | |
| Trainable params: 1,105,158 | | | |
| Non-trainable params: 0 | | | |

Figure 14: Model Summary of the End-to-end Agent

4.3.3 Training and Testing

We train our agent end-to-end on the training set with 10 cores and a batch size of 4096. We optimise our network using an adam optimiser based on categorical cross entropy loss. Since the dataset consists of all possible conversational points, we train our agent only for a single epoch. We report categorical cross entropy, categorical accuracy, precision and recall on the test set.

4.3.4 Results

The training takes roughly five hours on an `lrz.xlarge` instance. This process is done in two steps to mitigate memory issues that arise due to a memory leak in Keras implementation. The results on the test set are as follows (rounded to 3 digits after comma):

- Categorical Cross Entropy: 1.011
- Categorical Accuracy: 0.753
- Precision: 1.0
- Recall : 0.751

The sizable difference between the precision and the recall points towards a low number of false positives and a high number of false negatives. Considering the huge and imbalanced

action space for our model, where many actions are infrequently selected - such as the slot filling actions - and a few actions are very frequently selected - such as the utterance actions, particularly *utter_pass_turn* which occurs at every conversational turn -, these results indicate that the model is conservative when it comes to selecting low frequency actions, preferring to select the high frequency actions in case of uncertainty. This behaviour results in high average precision and low average recall since a single action has a minimal effect (around $\sim 1.7\%$) on the macro-average. As such, high precision and low recall of the infrequent actions dominate the results.

In order to try and mitigate this issue, we have removed the most frequent action *utter_pass_turn* from the action space, treating an utterance as implicitly passing the turn, and trained the network again. However, we observed no significant improvements on the test metrics, indicating that the agent was still prone to selecting the high frequency actions in case of uncertainty.

Another way to mitigate this issue could be to train the model on a stratified dataset where the action frequencies are balanced. However, we do not explore this approach due to time constraints.

4.4 Selection of one Baseline Approach

The key characteristics of the three approaches can be compared in table 1. The intent based agent and the end-to-end agent both yield good results, especially considering the latter models complexity due to the incorporation of the natural language understanding capability. Going forward, however, we only consider the intent based agent due to its better performance and ease of extendability. While the exclusion of NLU introduces some limitations, we believe that the model’s simplicity will allow us to focus on the implementation of an intrinsic motivation complex. Furthermore, extension into NLU remains possible, for example through incorporation of another model able to map utterances to user intents and slot values.

| criteria | intent based approach | multi component approach | end-to-end approach |
|------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| training method | reinforcement learning | reinforcement learning | supervised learning |
| allows for NLU | not yet | yes | yes |
| performance of base model | very good (success rate of 98%) | not sufficient | good (accuracy of 75%) |
| easy extendability | yes | yes | moderate |
| necessary training resources | 1-2h of training (with a normal CPU) | 1-2h of training (with a normal CPU) | 5h of training (using the LRZ cloud) |

Table 1: Comparison of baseline concepts

5 Concepts for Intrinsic Motivation

One of the major goals for this project was to build an agent, which acts based on an intrinsic motivation. Therefore, for the the second part of the project the focus has been on finding approaches to extend the baseline agents, such that the agent acts in an intrinsically motivated fashion by triggering the user throughout the conversation for example. As we have selected the intent based approach from all baseline approaches, the concepts for intrinsic motivation are based on this baseline agent.

5.1 Mood-model Based Intrinsic Motivation

5.1.1 Overall Concept

One of our approaches for the intrinsic motivation was to define the motivation of the agent to fulfill the goal of finding a drink as an extrinsic motivation (as his job as a barkeeper) and to describe the intrinsic motivation as the endeavor of the agent to make the user happy. Therefore we approached the intrinsic motivation by introducing a mood model, describing the user’s mood at each step of the conversation.

While we have first thought of incorporating the user mood in the state $s(t)$, which will be passed to the agent, ultimately we decided against this approach and here is why: Passing on the user mood to the agent as a part of the state would mean, that the agent would be aware of the user mood at each step of the conversation. Thinking of a chatbot in real life, this would require a way to identify the user mood at each step of the conversation, either with face recognition or by extracting the user mood with a sentiment analysis from the answer he provides. As many answers from the user are rather neutral in such a conversation, this is a hard problem.

To circumvent this problem we have therefore chosen a different approach, which focuses purely on the reward, which the agent gets as a feedback for each of his actions $aa(t)$.

The core of this idea was to extend our reward formulation by distinguishing between an extrinsic and an intrinsic reward (see eq. 16), also stated as a common method in a survey for intrinsic motivation in RL [22] The intrinsic reward is then based on the mood model, which will be explained in further detail in the next section. By encoding the user mood in the reward, which is just used for training the DQN agent, we will not pass the user mood directly to the agent but rather indirectly. With this approach the agent will develop an intrinsic motivation to make the user happy during the training phase. As he has learned intrinsically how to behave in order to make a user happy by kind of anticipating but not knowing his mood, he doesn’t have to extract the real user mood in the deployment phase, which is a significant advantage.

5.1.2 User Mood

The basis for the mood model developed in this project has been the PAD emotional state model of Russell. PAD refers to the 3 dimensions of this model - pleasure, arousal and dominance. [23] We have adapted this model for our mood model by reducing it to a two dimensional model keeping the pleasure and arousal dimension. As seen in figure 15 the arousal dimension corresponds to the user’s goal desire for our use case, meaning how thirsty he or she is, and the pleasure is the user’s happiness. In our mood model the

user gets from one mood state em_t to another state em_{t+1} in a discrete fashion like in a finite state machine. In total a user can have six different states. In the beginning of a conversation the user gets randomly assigned to one of the six mood states. During a conversation transitions are only possible on the pleasure dimension, but not on the goal desire dimension, since realistically a real user will not get from a thirsty state to a not thirsty state and the other way around during the length of one conversation.

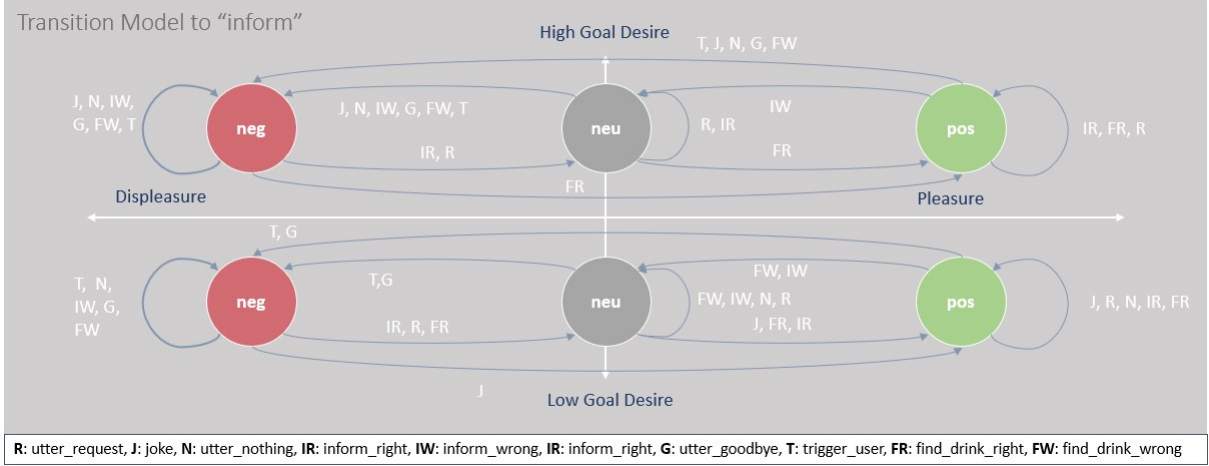


Figure 15: User mood model inspired by the PAD state model of Russell for the user utterance *inform*

The transitions for how a user changes from one mood state em_t to the next one em_{t+1} are defined individually for each user action and the corresponding agent actions as shown in figure 15 for the example of the user action *inform*. Each user action has its own mood model, with all the agent actions being the transitions from one state to another. Each state has a transition for each agent action.

The intrinsic reward function is based on these transitions between the mood states as the following:

$$r_{transition}(em_{t+1}|em_t) = r_{transition}(transition_{em_t-em_{t+1}}) = \begin{pmatrix} neg_neg & neg_neu & neg_pos \\ neu_neg & neu_neu & neu_pos \\ pos_neg & pos_neu & pos_pos \end{pmatrix} \quad (14)$$

Beyond the pure transitions the intrinsic reward also depends on where the user is on the goal desire dimension (see eq. 15). For a low goal desire it gets an additional positive reward r_{goal_desire} for each additional step, enforcing longer conversations, as the agent should learn to engage with the user longer, whenever he is not thirsty.

$$r_{mood} = r_{transition}(em_{t+1}|em_t) + \begin{cases} 0 & \text{if } goal_desire = high \\ r_{goal_desire} & \text{if } goal_desire = low \end{cases} = r_{intrinsic} \quad (15)$$

The overall reward with intrinsic (see eq. 15) and extrinsic reward (see eq. 11) can be computed as a weighted sum:

$$r = \beta_{reward} \cdot r_{intrinsic} + (1 - \beta_{reward}) \cdot r_{extrinsic} \quad (16)$$

The exact values of the rewards as well as the weighting factors of intrinsic and extrinsic reward have to be adjusted by experimentation and are therefore part of the hyperparameter tuning in chapter 6.

5.1.3 Additional Extensions to the Baseline Intent Based Agent

To adapt the user simulator and the agent to our concept of intrinsic motivation we have made several changes. As a first step we have added two new user utterances *nothing* and *nohelpful* and three new agent utterances *joke*, *trigger_user* and *nothing* to the original utterances from figure 3. The description of all intents can be in table 2 These new actions allow to have moments of silence in a conversation and give the agent the possibility to trigger the user or tell a joke on its own.

Beyond that we have made additional changes to the user simulator. User answers have been modified to be also dependent on the user’s goal desire mood and have been changed to be more stochastic and no longer only rule based. This makes the user simulator more complex, which is good for the agent to learn a more robust behavior. Furthermore we have changed the termination condition of an episode. As explained in 4.1.1 so far the user has closed the conversation after a maximum number of steps and whenever this happened, the agent has received a negative reward - independently of whether he had found a drink before. This behavior enforced the agent to close the conversation right after he had found a drink - leading to a very efficient conversation. However with the additional intrinsic goal of making the user happy, we still want to fulfill our goal of finding the drink, but we also want to lead a conversation, which makes the user happy beyond the drink finding. Therefore we have changed the termination condition such that the user is still closing the conversation after a maximum number of steps, but that the agent still gets a positive reward, when he has found a drink throughout the conversation. Additionally to that, the user now also closes the conversation, whenever he is in a negative mood for the third time giving a high negative reward even when a drink has been found. This increases the importance for the agent to keep the user in a positive mood.

5.2 Curiosity-driven Intrinsic Motivation

5.2.1 Motivation

A huge difficulty in reinforcement learning scenarios is the efficient exploration of state-action pairs through interactions with the environment by the agent and a good trade off between exploration and exploitation as mentioned in section 3.3. A common used method is ϵ - greedy exploration, where the agent selects a random action with a probability of ϵ . With a probability of $1 - \epsilon$ the agent selects the actions that maximizes its Q-value function as explained in 4.1.2. In order to converge to an optimal value, ϵ has to converge to zero with increasing number of episodes assuming enough knowledge is collected over

time. This strategy is quite inefficient and often does not lead to the desired goal, as it neither takes the difficulty of a certain skill into account nor the order in which skills are learned. Methods of intrinsic motivation such as curiosity can tackle this issue.

Developmental psychologists describe curiosity as a form of intrinsic motivation that is the primary driver in the early stages of development in humans [24]: especially babies appear to employ goal-less exploration to learn skills that will be useful later on in life.

As we employed ϵ -greedy exploration for the intent based baseline 4.1, we concluded that the implementation of a curiosity driven learning/exploration approach is advantageous for our training. It also aligns with the overall goal of our project to construct an intrinsic motivated agent and would even work with the intrinsic mood concept presented in 5.1.

5.2.2 Implementation

We chose to implement the curiosity-driven exploration by self-supervised prediction approach by Pathak et al. [7] as it is one of the most influential approaches and the concept is well described. The general idea is the following:

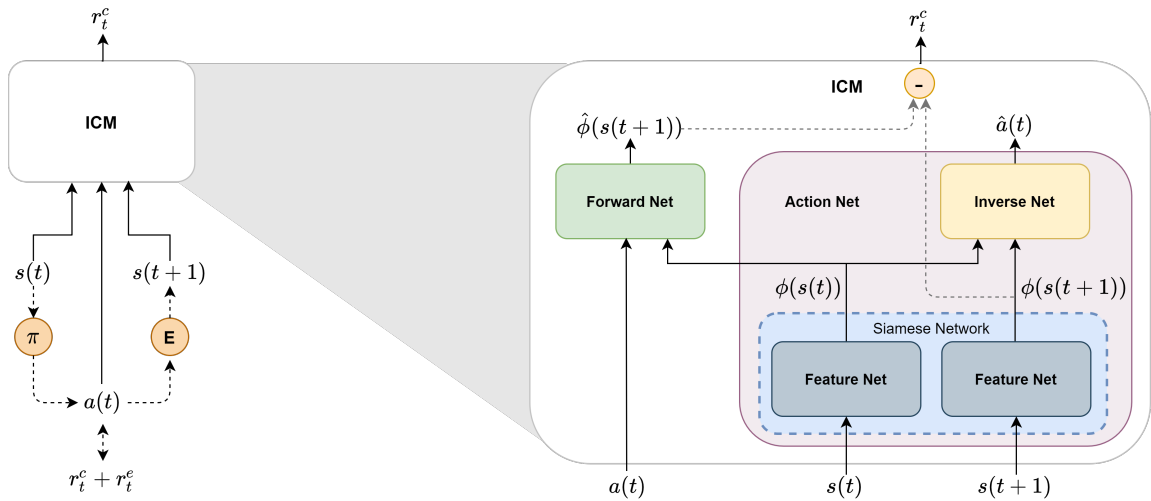


Figure 16: Schematic architecture of the intrinsic curiosity module

Consider an agent that gets its state $s(t)$ from the state tracker, takes an action based on its current policy π and transitions to the next state $s(t+1)$ based on its interaction with the environment \mathbf{E} . This process is depicted in figure 16 on the left side. The idea is to incentivize the agent with a reward r_t^c describing how informative this transition was. We will call this incentive reward from now on curiosity reward. It is calculated by the intrinsic curiosity module (ICM). The schematic architecture of the ICM is depicted in 16 on the right side. Generally the ICM consists of two neural networks - the action net and the forward net. The action net in return consists of two networks: a feature network that embeds a state vector into feature representation $\phi(s(t))$ or $\phi(s(t+1))$ and an inverse net that calculates the inverse dynamics of the agent by predicting the action $a(t)$ that connects $s(t)$ and $s(t+1)$ based on the feature representation of $\phi(s(t))$ and $\phi(s(t+1))$. The forward net predicts the feature representation $\hat{\phi}(s(t+1))$ by exploiting the current action $a(t)$ and the state feature representation $\phi(s(t))$ calculated by the

feature net. Details about the architectures of the neural networks can be found in table 4. The curiosity reward is then calculated by:

$$r_t^c = \|\hat{\phi}(s(t+1)) - \phi(s(t+1))\|^2 \quad (17)$$

The reason why the concept is called curiosity-driven exploration by self-supervised prediction lays in the fact that both networks, the action network and forward network can be trained efficiently with samples obtained by interactions between the agent and the environment. That means that both network can be trained with the same samples $(s(t), a(t), r(t), s(t+1))$ as the agent. The action network is trained by predicting the action $\hat{a}(t) = g(s(t), s(t+1); \theta_{action})$ and by minimizing the loss by minimizing the multi class cross entropy of the output layer of the inverse net:

$$\min_{\theta_{action}} L_{action}(\hat{a}(t), a(t)) \quad (18)$$

We point out that the feature net is implemented as a siamese network architecture, which ensures that the network shares the same weights for both inputs $s(t)$ and $s(t+1)$. The forward network is trained by predicting the feature representation $\hat{\phi}(s(t)) = f(a(t), \phi(s(t)); \theta_{forward})$ and by minimizing the loss:

$$\min_{\theta_{forward}} L_{action}(\hat{\phi}(s(t)), \phi(s(t))). \quad (19)$$

As for the formulation of the curiosity reward r_t^c , the loss is calculated by taking the mean squared error between the prediction and the ground truth. This leads us to the combined optimization problem of the entire training process additionally incorporating the optimization of the DQN Agent:

$$\min_{\theta_{action}, \theta_{forward}, \theta_{agent}} -L_n(r_{overall}(\lambda)) + \beta_{curiosity} \cdot L_{forward} + (1 - \beta_{curiosity}) \cdot L_{action} \quad (20)$$

The speciality about this optimization problem is its antagonistic nature. While the agent is trained to maximize its overall reward $r_{overall}(\lambda)$ including the curiosity reward, the forward net tries to minimize the curiosity reward. That means the agent will favor actions with high prediction error, which will be higher in areas where the agent has spent less time or anytime at all, or in areas of more complex tasks. With time evolving, the forward net incorporates already experienced interactions with the environment and the curiosity decreases. The action net is needed to construct a feature space ϕ that encodes only those features of the state and therefore of the environment, which can be influenced by the agent.

Due to the antagonistic nature of this concept the training becomes more complicated, as neither the agent nor the forward net should dominate the curiosity reward/loss. Additional to that more hyperparameters are introduced such as λ and $\beta_{curiosity}$ and the specific architectures of the action net and forward net implemented as multilayer perceptrons.

The reward formulation of the overall reward $r_{overall}$ described in eq. 16 is extended with the curiosity reward r_c as follows:

$$r_{intrinsic} = \alpha_{reward} \cdot r_{mood} + (1 - \alpha_{reward}) \cdot \lambda \cdot r_{curiosity} \quad (21)$$

$$r_{overall} = \beta_{reward} \cdot r_{intrinsic} + (1 - \beta_{reward}) \cdot r_{extrinsic}, \quad (22)$$

where $\alpha_{reward} \in [0, 1]$, $\beta_{reward} \in [0, 1]$ and $\lambda \in \mathbb{R}^+$

6 Experiments

6.1 Metrics

To evaluate the model’s performance during different experiments we first had to construct a suitable metric. According to [25] there are two main aspects to be evaluated when using task-oriented dialogue systems: task success and dialogue efficiency. Two metrics to assess the task success and one metric for the dialogue efficiency have been used. Additionally a combination of the success and intrinsic mood reward has been used to evaluate the model performance. An overview of these four metrics is given in figure 17. In order to combine all four metrics into one number a quality metric has been developed.

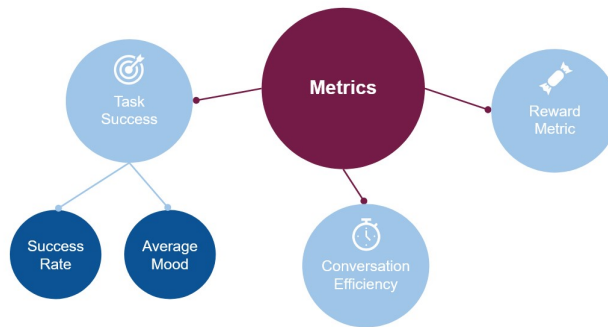


Figure 17: Overview of the Metrics

6.1.1 Task Success

In order to measure the task success we have used two metrics: success rate and average mood. The agent acts successfully if the user is happy and the user goal is fulfilled. The success rate describes the fraction of conversation in which the agent managed to find the correct drink:

$$M_{success} = \frac{| \text{conversations in which the user's goal is met} |}{| \text{all conversations} |} \quad (23)$$

This metric was already used for the baseline agent and could reach a value of 98%. Average mood describes the average mood of the user simulator over all conversation steps, where 0, 0.5 and 1 correspond to a negative, neutral and positive mood respectively.

$$M_{mood} = \frac{\sum_{i=1}^{max_round} mood_{user}(i)}{max_round} \quad (24)$$

where

$$mood_{user} = \begin{cases} 0 & \text{if mood is negative} \\ 0.5 & \text{if mood is neutral} \\ 1 & \text{if mood is positive} \end{cases} \quad (25)$$

6.1.2 Dialogue Efficiency

With the aim of measuring the efficiency of a dialogue one has to differentiate between the goal desires of a user. If the goal desire is high and the agent takes every agent action (differentiating between different request and inform slots) at most once the conversation is considered to be efficient. If the goal desire is low and the agent will take every agent action at most once apart from utter_nothing and joke, which it can take arbitrarily often, the conversation is considered to be efficient. This ensures that the user is not confronted with repeating requests or informs, but can still have a long conversation with the agent if the user’s goal desire is low. The formula for measuring the efficiency of the conversation is

$$M_{eff} = \begin{cases} \frac{|unique\ agent\ actions|}{|agent\ actions|}, & \text{if goal desire = high} \\ \frac{|unique\ agent\ actions|}{|agent\ actions^*|}, & \text{if goal desire = low} \end{cases} \quad (26)$$

where $|agent\ actions^*|$ corresponds to the number of actions taken by the agent counting the actions utter_nothing and joke only once.

6.1.3 Reward Metric

Since the total reward clearly depends on the specific choice of reward hyperparameters described in 2, we introduce the reward metric M_{rew} , which allows for better comparability in the hyperparameter space. We achieve this by focusing only on the goal reward r_{goal} defined as:

$$r_{goal} = \alpha_{reward} \cdot \beta_{reward} \cdot r_{mood} + (1 - \alpha_{reward}) \cdot r_{ext}. \quad (27)$$

Not considering the curiosity reward allows to only take into account the reward related to the conversation, as the the curiosity reward is a reward that incentivises the exploration during training. Therefore, it doesn’t resemble the quality of the conversation. In order to get M_{rew} , we normalize r_{goal} with the maximal achievable reward $r_{goalmax}$, such that:

$$M_{rew} = \frac{r_{goal}}{r_{goalmax}} \quad (28)$$

$r_{goalmax}$ is defined as:

$$r_{goalmax} = \alpha_{reward} \cdot \beta_{reward} \cdot r_{maxmood}^{total} + (1 - \alpha_{reward}) \cdot r_{maxext}^{total}, \quad (29)$$

where $r_{maxmood}^{total}$ is defined as:

$$r_{maxmood}^{total} = (\max(r_{transition}) + r_{goal\ desire}) \cdot \max_round \quad (30)$$

and r_{maxext}^{total} is defined as:

$$r_{maxext}^{total} = (w_{success} + r_{step}) \cdot \max_round, \quad (31)$$

with $w_{success}$ being the hyperparameter to weight the success if a goal is found.

6.1.4 Quality Metric

In order to combine the four metrics described above the formula

$$M_{quality} = \alpha_{met}M_{eff} + \beta_{met}M_{rew} + \gamma_{met}M_{mood} + \delta_{met}M_{success} \quad (32)$$

is used, where α_{met} , β_{met} , γ_{met} and δ_{met} sum up to one and are therefore weighting factors. $M_{success}$, M_{mood} , M_{eff} and M_{rew} correspond to the success, average mood, efficiency and reward metric, respectively. Since we consider the success of the conversation to be the most important indicator and the rest of the metrics are equally important for evaluating the performance of the model we set δ_{met} to 0.4 and the other parameters to 0.2.

6.2 Hyperparameter Experiments

With the metrics described in section 6.1, we were able to perform an automatized large scale hyperparameter search in the end. A detailed description of the most important hyperparameters is given in table 2. The set of hyperparameters is of dimension 30, as there are also hyperparameters that consist of multiple parameters such as $r_{transitions}$, which consists of 9 specific values. However, a dimensionality of 30 leads to a complex search problem. Therefore, we simplified the problem by dividing the hyper parameter set to into four subsets. On each of the subsets we performed grid search on specified ranges of values that we found during the implementation of concepts and their validation.

The first subset comprises 13 hyperparameters describing the values of the rewards $r_{transitions}$, r_{goal_desire} , r_{step} , $w_{success}$ and w_{fail} . As simplification we only considered 3 different sets for $r_{transition}$ that are the most reasonable.

The second subset comprises three hyperparameters describing the architecture of the neural networks and the curiosity optimization problem: dqn_hidden_size , $feature_size$ and $\beta_{curiosity}$. The third subset comprises three hyperparameters describing the combination of different rewards used in equations 21 and 22: α_{reward} , β_{reward} and λ . The last group comprises 11 hyperparameters that proved good performance before and were not optimized: $flush$, $learning_rate$, $extra_punishment$, $batch_size$, max_round_number , $warm_up_number$, γ_{reward} , $replay_size$, $emc_{slot_error_mode}$, $emc_{slot_error_prob}$ and $emc_{intent_error_prob}$. Each individual hyperparameter combination was trained for 3000 episodes which took around 90 minutes on the LRZ cloud.

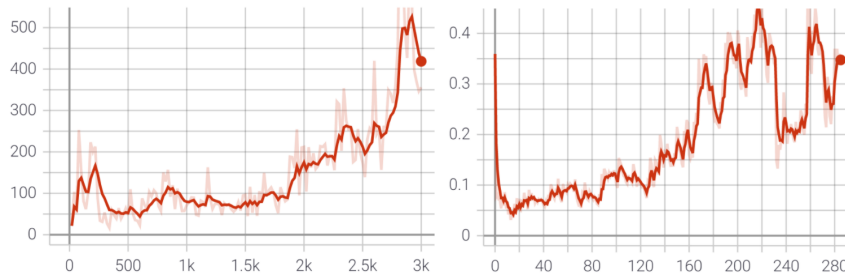


Figure 18: Example of diverging curiosity reward with the curiosity reward plotted against the episode number (left) and the loss of the forward net plotted against the training cycles (right)

6.2.1 Intermediate Results

After the first 120 training combinations, intermediate results were analyzed. Two main observations could be drawn: The curiosity reward is very sensitive to the hyperparameters and can diverge quickly. This happened in all cases but especially for large values of $\lambda > 0.3$ and $\alpha_{reward} < 0.6$. Such a training situation is exemplary depicted in figure 18. Translated to the RL problem, this means, that the agent learns much faster how to maximize its curiosity reward than the forward net to minimize its prediction error based on its "experience". Therefore, the training strategy of the ICM was changed. While the agent was trained on a certain number of batches for one epoch, the ICM was trained for two epochs on the same data. Another explanation was that the forward net was limited by its size and depth to predict the correct feature representation of $s(t + 1)$. Therefore, in order to increase the capacity of the network, the depth was extended by two additional hidden layers, and the size of the layers was increased. Both measures led to the stabilization of forward nets loss. It was observed that the reward metric was not a metric that allows to compare the quality of different models. Despite being designed to range between 0 and 1 for any hyperparameter combination, it doesn't take into account that for low values of β_{reward} it is a lot easier for the agent to reach its maximal reward, as the rewards for the mood are barely taken into account and are a lot more difficult to reach. As a consequence the quality metric was adapted to not taking the reward metric into account. With $\alpha_{met} = 0.3, \gamma_{met} = 0.3$ and $\delta_{met} = 0.4$. this changes equation 32 to:

$$M_{quality} = \alpha_{met}M_{eff} + \gamma_{met}M_{mood} + \delta_{met}M_{success} \quad (33)$$

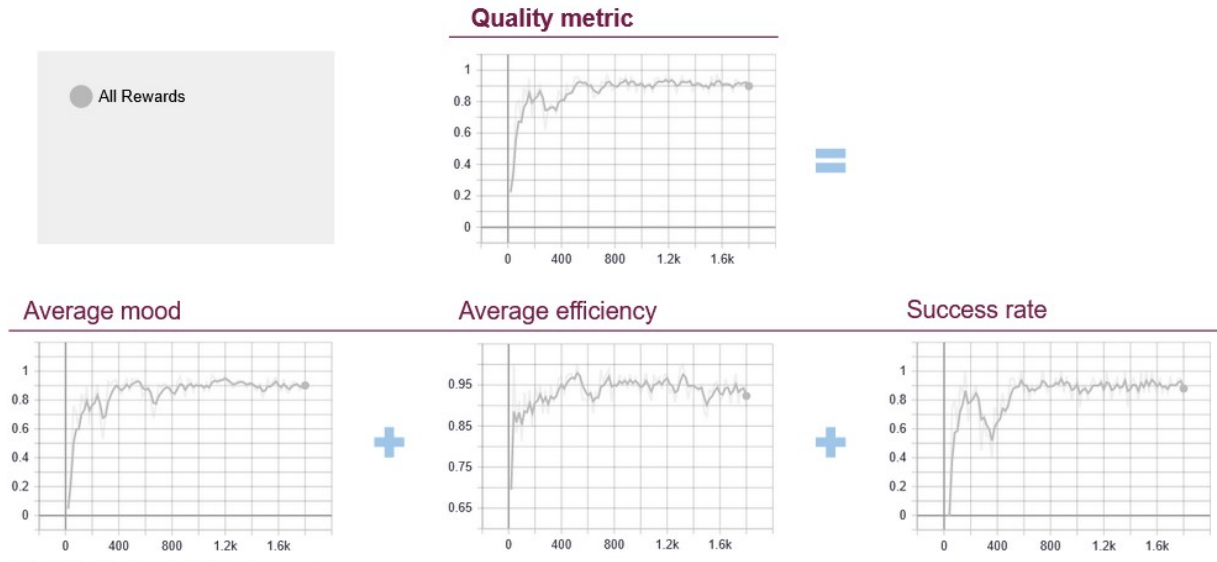


Figure 19: The quality metric of the the best hyperparameters and related constituents

6.3 Final Results

After performing the aforementioned adaptations further training runs were conducted for 1800 episodes and the best hyperparameters combined. The best result is depicted in

figure 19, showing the quality metric and its constituents. The best hyperparameter values can be found in 2.

As the metric is an abstract way to determine the quality of a conversation, we examined the conversations of the chatbot with the best hyperparameters by hand. An excerpt of the conversation can be found in figures 26 and 25. In figure 26, the user has a high goal desire. The agent recognizes this and chooses the actions that lead to the goal as quickly as possible. In figure 25, the user has a low goal desire and starts the conversation by saying *nothing*. Then the agent correctly triggers the user to order a drink. He tells a joke to make the user happy and in some situations says *nothing* because there is no urgency. A joke doesn't always lead to a positive mood, as in phases where silence is desired. In the end, the agent finds the right drink. In summary, the agent has learned the following skills that improve the baseline.

1. Distinguishing between high and low goal desire of the user
2. Triggering the user to interrupt silence
3. Not saying anything when silence is appreciated
4. Telling jokes to enhance the users mood
5. Finding a drink even if wrong information was given by the user

The main result of the hyper parameter tuning experiment is that we found a set of reasonable good performing hyperparameters which allowed to realise the abilities shown above. Since the set of all valid hyperparameters is infinitely large, it is not possible to find the optimal configuration through exhaustive search. As such, while there might be better performing hyperparameters, we do not further search for any superior configuration. We find the performance of the current set of hyperparameters to be satisfactory with regards to the goal of the experiments, which is to demonstrate the validity of the intrinsic motivation concept.

6.4 Comparison Intrinsic Motivation Concepts

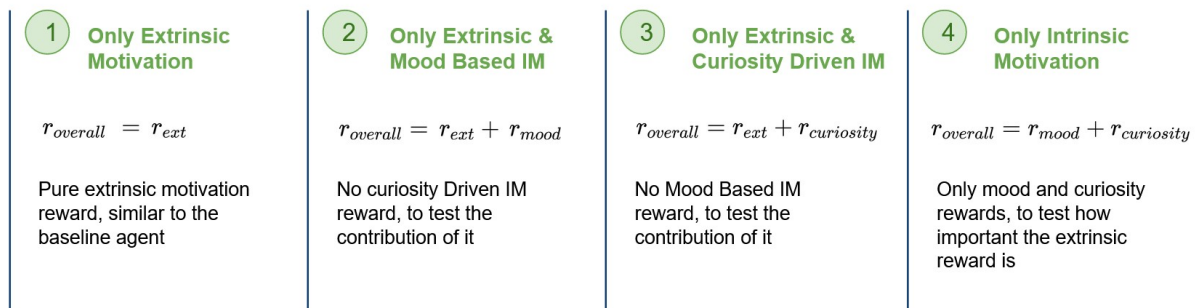


Figure 20: Overview of 4 different ablation studies

After we found a good model with the hyperparameter tuning we wanted to validate, whether our two concepts for intrinsic motivation truly improve the chatbot and how

they influence the agent’s behavior. For this sake we conducted four different ablation studies (see fig. 20).

6.4.1 Comparison Metrics

The result graphs 21 show the quality metrics and its four sub metrics. Looking at the average mood metric, which gives the best indication about the quality of the conversation regarding the user’s happiness, the experiment with only extrinsic and mood based motivation (blue model) performs best. Secondly, for the average efficiency metric the experiment with only extrinsic motivation (red model), and the one combining all rewards (grey model) perform best, which means that in the others we have a lot of repetitions in the conversation.

The success rate indicating how often the goal has been reached, shows that the experiment of only intrinsic motivation and the experiment without mood based intrinsic motivation perform worse than the other. This shows on one hand that it is very hard for the agent to find a drink without any goal reward from the extrinsic motivation. On the other hand it shows that combining the extrinsic goal reward with a curiosity reward distracts the agent more from finding a goal, than having either a pure extrinsic goal reward or an extrinsic goal reward combined with a mood reward. This can be attributed to the fact that the curiosity reward motivates the agent to explore more, which can also distract him to find a drink, if it is not guided well.

It can be assumed that this is the reason why the combination of all three motivation rewards gives the best conversations overall as shown in the overall quality metric. Combining all three rewards, we not only introduce this exploration behavior leading to a more versatile conversation, but also guide the agent with the dense mood reward. The very high extrinsic reward, then rewards the agent additionally if the goal is found in the end, enforcing overall a versatile conversation, which still leads to a goal finding.



Figure 21: Comparison of the Quality Metrics and its 4 submetrics for all ablation studies

6.4.2 Analysis of Conversation Examples

After looking at the graphs, we also investigated some conversations for the different experiments. The results of this analysis are summarized in figure 22. While for the first experiment, we saw good metric graphs, we recognized that the agent can not distinguish between high and goal desire anymore, which only leads to short and efficient conversations without pauses and jokes. The second experiment shows a good distinction between low and high goal desire, with jokes and pauses but also many repetitions. The third experiment shows that the agent does not learn to find a goal at all as he is exploring too much and only tells jokes and says nothing for a low goal desire, while slightly performing better for a high goal desire. The fourth experiment shows that without a goal reward the agent does not finish a conversation after having found a drink and often does not care to find a drink at all leading to very long conversations mainly consisting of the agent actions *joke* and *utter_nothing*.

Therefore, the ablation study shows that excluding any of the rewards in our model leads to worse results than having a combination of all three rewards, which validates our two intrinsic motivation concepts.

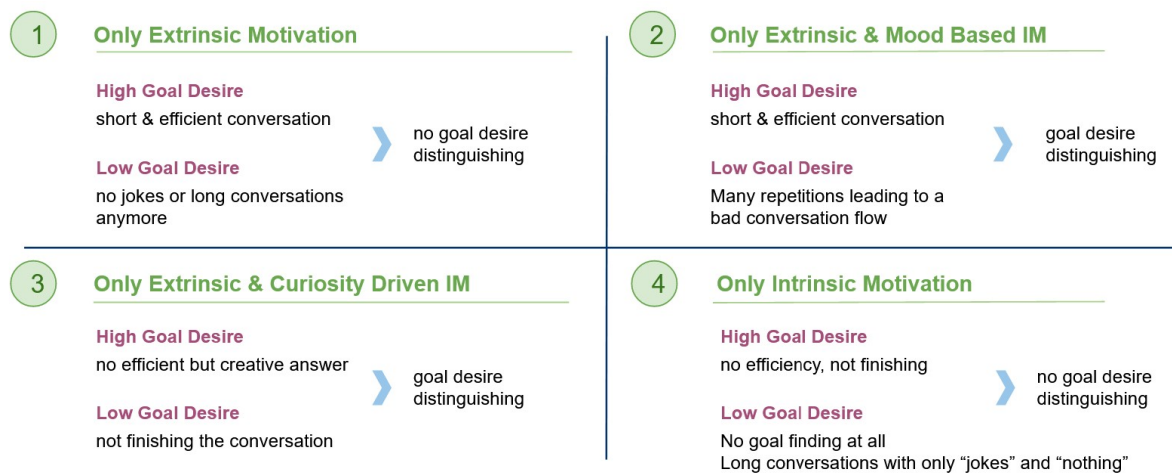


Figure 22: Overview of the conversation quality for all ablation studies

6.5 Continuous skill expansion

To check the extendibility of our intent-based agent we examined the agent's behavior by introducing a new slot during test time. During training time, the agent was trained using partially filled slots. The user goals used for training were also modified so that they only contain information about drink names and size slots. During test time information for the temperature slot was added to the user goals. The agent was not able to generalize as it did not ask the user for the temperature of the drink in figure 6.7 before confirming the drink as it did in figure 6.8. However, it was able to determine the other actions at the expected time. Hence, its previous knowledge remained intact and unaffected on encountering some new data. Moreover, the agent with partially filled slots optimized its performance in about 260 episodes while for an agent trained from scratch, it took about 500 episodes to optimize.

```

*****Episode 4 *****
User Goal: {'request_slots': {'drinknumber': 'UNK'}, 'dialect': 'request', 'inform_slots': {'DRINK': 'barracuda', 'SIZE': 'medium', 'TEMP': 'cold'}}
{'goal_desire': 'high', 'emotion': 'neutral'}
-----
Initial User Utterance: Can I have a medium drink
Agent Action: Which DRINK do you like?
***User Response: I would like this to be barracuda
---User mood:neutral
Agent Action: We have the following drink: barracuda medium cold 28
***User Response: Thanks. I will take this :)
---User mood:positiv
Agent Action: bye bye
***User Response: goodbye
---User mood:positiv

```

Figure 23: A sample of agent’s conversation trained with partially filled slots

```

*****Episode 43 *****
User Goal: {'request_slots': {'SIZE': 'UNK', 'drinknumber': 'UNK'}, 'dialect': 'request', 'inform_slots': {'TEMP': 'cold', 'DRINK': 'white vine'}}
{'goal_desire': 'high', 'emotion': 'neutral'}
-----
Initial User Utterance: {'intent': 'order_drinks', 'request_slots': {'SIZE': 'UNK'}, 'inform_slots': {'TEMP': 'cold', 'round': 0, 'speaker': 'User'}}
Agent Action: Which DRINK do you like?
***User Response: I would like this to be white vine
---User mood:neutral
Agent Action: Which SIZE do you like?
***User Response: Which SIZE do you have?
---User mood:negativ
Agent Action: Which TEMP do you like?
***User Response: I would like this to be cold
---User mood:negativ
Agent Action: We have the following drink: white vine small cold 39
***User Response: Thanks. I will take this :)
---User mood:positiv
Agent Action: bye bye
***User Response: goodbye
---User mood:positiv

```

Figure 24: A sample of agent’s conversation trained with filled slots

7 Conclusion and Future Work

The goal of this project was to address two issues that are present in all assistive chatbots to date: the inability to trigger actions autonomously if necessary, and the restricted skill expansion over time. Therefore the project aimed to develop a concept for intrinsic motivation for a goal-oriented chatbot such that it aligns with the human notion for intrinsic motivation.

The baseline requirements were that our chatbot is able to carry on coherent conversations and aid the user in achieving his goals, as this is the basic purpose of such an agent. Three baseline agents have been developed, one of which has been chosen to be extended with intrinsic motivation modules to address the targeted issues.

The intrinsic motivation module supplements the extrinsic motivation of the agent to fulfil the user goal with the intrinsic motivation to improve the user's mood. To achieve this, a mood model based on Mehrabian and Russell's PAD emotional state model [23] has been developed for the user, where the agent is rewarded depending on the user's goal desire and changes in the user's happiness. This is implemented via an intrinsic reward function, which is initially separate from the extrinsic reward function. This separation of extrinsic and intrinsic goals allows the agent to learn to trigger actions independently from explicit goals, as the implicit goal is the improvement of the user mood, which cannot be improved without initiating an action.

The ability of the agent to continuously explore skills is implemented via a curiosity module. It consists of two antagonistic networks, where on the one hand the agent tries to maximize the overall reward (including a curiosity reward), while the forward net on the other hand tries to minimize the curiosity reward, which leads to the agent favouring more complex or unknown actions. This tug-of-war rewards both informational gain and goal completion.

All modules (baseline, intrinsic motivation and curiosity) can be combined, by integrating the respective reward functions into an overall reward. This can be compared to human neurophysiology where several motivational processes share partially identical reward systems [5].

In the end the baseline agent could be supplemented with valuable abilities including the ability to trigger the user to interrupt silence. Furthermore, our experiments could show that the developed modules significantly increase different performance measures of the agent, whereas a combination of all modules was shown to be the most successful. This validates the approaches and implementations we have developed and shows them to be a valid basis for further developments. However, we point out that our environment describes a rather specific use case and we believe that even more benefits of the curiosity model can be seen in a more complex environment as it will help the agent in exploration and also generalizing. The fact that we implemented the motivation modules using reinforcement learning means, that there is great potential for flexible adaptation of the agent to multiple different domains with varying numbers of skills. It would also be possible to introduce a general purpose mood detection module with a natural language understanding capability that infers the user mood directly from the user utterance and the current state. Furthermore, based on our experiences in this project we put stress on the importance of a well-designed user simulation as an asset in developing modules/concepts which are dependent on the agent-user-interaction and to enable reliable training of

the bot and the implemented extensions. For this reason we consider the User Simulator a primary key to success with a DRL Chatbot. However, the task of building a good user simulator is as difficult as coming up with a good agent if not more. Therefore we have also implemented a second user environment, with which real users can train the agent through manual inputs, to finetune it with more versatile and arbitrary answers, than the rule-based user simulator's answers.

The next steps to build on this body of work include the further optimization of the hyperparameters and research on the extension of our concepts to a broader range of settings. Furthermore, developments in the curiosity module to improve the agent's ability to generalize are possible. As we have chosen to build the prototypes of the motivation modules based on the intent-based baseline agent, revisiting NLU approaches will help to achieve this.

All in all, our project was successful in providing a proof of concept and lays the foundations for further refinement and expansion of intrinsic motivation concepts in chatbots. While these concepts are still in their infancy, we have contributed to the advancement of chatbots in terms of the acquisition of new skills and the more flexible interaction with the user and his goals. Together with future supplementations to our project this use-oriented research can make a noticeable change in our interaction and use of chatbots.

A Appendix

| Network | Layers | Size |
|--------------------|------------------------------------|-----------|
| dqn agent | linear + ReLU | 200 x 140 |
| | linear + ReLU | 140 x 100 |
| | linear + ReLU | 100 x 80 |
| | linear | 80 x 13 |
| action net | | |
| feature net | linear + ReLU | 200 x 140 |
| | linear + ReLU | 140 x 140 |
| | linear + ReLU | 140 x 80 |
| inverse net | linear + ReLU | 160 x 140 |
| | linear + ReLU | 140 x 140 |
| | linear | 140 x 13 |
| forward net | linear + ReLU + dropout (p=0.5) | 93 x 280 |
| | linear + ReLU | 280 x 280 |
| | linear + ReLU | 280x 280 |
| | linear | 280 x 80 |


Table 3: Neural network architectures

| Hyperparameter | Description | Best Value(s) |
|------------------------------|--|---|
| <i>max_round</i> | maximum number of rounds per conversation | 30 |
| <i>flush</i> | determines whether to flush the replay buffer after a certain number of episodes | 120 |
| <i>learning_rate</i> | learning rate for the dqn agent | 10^{-3} |
| <i>batch_size</i> | batchsize to train the dqn agent on | 64 |
| <i>warm_up_number</i> | Number of samples to get rule based actions from the agent to initially fill the replay buffer | 2000 |
| <i>w_fail</i> | weighting factor of extrinsic reward in case of a negative success | -1 |
| <i>w_success</i> | weighting factor of extrinsic reward in case of a success | 2 |
| <i>r_step</i> | extrinsic reward for one conversation step | 0 |
| <i>extra_punishment</i> | additional punishment for the agent if the exact same drink is found more than once | 10 |
| <i>r_transition</i> | reward matrix based on mood transitions with values between -10 and 10 for each matrix entry | $\begin{pmatrix} -5 & 1 & 10 \\ -7 & 0 & 10 \\ -10 & -1 & 10 \end{pmatrix}$ |
| <i>r_goal_desire</i> | additional mood reward for every conversation step in case of a low goal desire | 2 |
| β_{reward} | weighting factor to combine intrinsic and extrinsic reward to one reward value | 0.64 |
| α_{reward} | weighting factor to combine mood and curiosity reward to one intrinsic reward value | 0.75 |
| λ | factor to weight curiosity reward | 0.8 |
| <i>dqn_hidden_size</i> | basic size of a linear layer for the dqn agent | 130 |
| <i>feature_size</i> | size of the state encoding layer in the feature net and forward net | 80 |
| $\beta_{curiosity}$ | factor weighting the loss between forward net and action net | 0.5 |
| γ_{reward} | discount factor | 0.9 |
| <i>replay_size</i> | memory size in number of samples for the replay buffer of the dqn agent | 50000 |
| <i>emC_slot_error_mode</i> | mode of the slot error (0 for replacing only the slot value, 1 for slot and its value and 2 in order to delete the slot) | 0 |
| <i>emC_slot_error_prob</i> | probability with which the slot error is caused | 0.05 |
| <i>emC_intent_error_prob</i> | probability with which a intent error is caused | 0 |

Table 2: Overview of Hyperparameters


| Intent | Description | Speaker |
|---------------|--|----------------|
| utter_request | Requests a value for a slot, which has currently no value | agent |
| utter_inform | Informs a slot, filled with a specific value | agent |
| find_drink | Proposes a drink, with all inform slots filled with values or 'no match available' if no drink has been found | agent |
| utter_goodbye | Closes the conversation, which triggers the user to check, whether the goal has been reached throughout the conversation | agent |
| order_drinks | Requests and informs certain slots in the beginning of a conversation | user |
| inform | Informs a slot, filled with a specific value | user |
| request | Requests a value for a slot, which has currently no value | user |
| reject | Rejects the drink, if it doesn't match the slot values from the goal inform slots | user |
| thanks | Accepts the drink, if it does match the slot values from the goal inform slots | user |
| goodbye | Closes the conversation either after the maximum number of steps has been reached, it is in a negative mood for the third time, or the agent closes the conversation | user |
| nothing | Says nothing, which is equivalent to a time step without utterance | user |
| trigger_user | Triggers the user, by asking him which drink he could offer the user. | user |
| joke | Makes a joke. Currently only one type of joke is implemented. | agent |
| utter_nothing | Says nothing, which is equivalent to a time step without utterance | agent |
| not_helpful | Indicates, that the agent requested or informed something wrong | user |

Table 4: User and agent intents



low goal
desire

Inform: medium, white vine **Request: TEMP**



```

Initial User Utterance: ...
Agent Action: Hey, sorry to disturb you. What DRINK can I offer to you?
***User Response: I would like this to be white vine
--User mood:positiv
Agent Action: Helvetica and Times New Roman walk into a bar. Get out of here! shouts the bartender. We don't serve your type.
***User Response: ...
--User mood:positiv
Agent Action: ...
***User Response: Which TEMP do you have?
--User mood:positiv
Agent Action: We have cold
***User Response: I would like this to be medium
--User mood:positiv
Agent Action: ...
***User Response: ...
--User mood:positiv
Agent Action: Helvetica and Times New Roman walk into a bar. Get out of here! shouts the bartender. We don't serve your type.
***User Response: ...
--User mood:neutral
Agent Action: ...
***User Response: ...
--User mood:neutral
Agent Action: Helvetica and Times New Roman walk into a bar. Get out of here! shouts the bartender. We don't serve your type.
***User Response: ...
--User mood:positiv
Agent Action: ...
***User Response: ...
--User mood:positiv
Agent Action: ...
***User Response: ...
--User mood:positiv
Agent Action: We have the following drink: white vine medium cold 40
***User Response: Thanks. I will take this :)
--User mood:positiv
Agent Action: bye bye
***User Response: goodbye
--User mood:positiv
Episode: 1 Success: True Reward: 63.91999999999999

```

Figure 25: Example conversation with the best hyperparameters with a low goal desire



high goal
desire

Inform: tea **Request: TEMP, SIZE**



```

Initial User Utterance: Can I have a tea drinkWhich TEMP do you have?
Agent Action: We have cold
***User Response: Which SIZE do you have?
--User mood:positiv
Agent Action: We have the following drink: tea small cold 19
***User Response: Thanks. I will take this :)
--User mood:positiv
Agent Action: bye bye
***User Response: goodbye
--User mood:positiv
Episode: 7 Success: True Reward: 26.400000000000006

```

Figure 26: Example conversation with the best hyperparameters with a high goal desire

References

- [1] F. Rheinberg, “Intrinsic motivation and flow,” in *Motivation and Action*, J. Heckhausen and H. Heckhausen, Eds. Cambridge: Cambridge University Press, 2008, pp. 579–622.
- [2] J. Eccles, “Expectancies, values and academic behaviors,” in *Achievement and achievement motives: Psychological and sociological approaches*, J. T. Spence, Ed. San Francisco, CA: W.H.Freeman Co Ltd, 1983, pp. 75–146.
- [3] J. Heckhausen and H. Heckhausen, “Motivation and action: Introduction and overview,” in *Motivation and Action*, J. Heckhausen and H. Heckhausen, Eds. Cambridge: Cambridge University Press, 2008, pp. 1–9.
- [4] M. Wiering and M. van Otterlo, “Reinforcement learning. state-of-the-art,” *ArXiv*, 2012. [Online]. Available: <http://dx.doi.org/10.1007/978-3-642-27645-3>
- [5] R. L. Cervera, M. Z. Wang, and B. Hayden, “Curiosity from the perspective of systems neuroscience,” *PsyArXiv*, 2020. [Online]. Available: <https://psyarxiv.com/znrbf/>
- [6] M. Eppe, C. Gumbsch, M. Kerzel, P. D. Nguyen, M. V. Butz, and S. Wermter, “Hierarchical principles of embodied reinforcement learning: A review,” *ArXiv*, 2020. [Online]. Available: <http://arxiv.org/pdf/2012.10147v1>
- [7] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 488–489, 2017.
- [8] J. Beckmann and H. Heckhausen, “Motivation as a function of expectancy and incentive,” in *Motivation and Action*, J. Heckhausen and H. Heckhausen, Eds. Cambridge: Cambridge University Press, 2008, pp. 99–132.
- [9] A. H. Maslow, “A theory of human motivation,” *Psychological Review*, vol. 50, no. 4, pp. 370–369, 1943.
- [10] P. Woergoetter, “Reinforcement learning.” [Online]. Available: http://www.scholarpedia.org/article/Reinforcement_learning
- [11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 1998.
- [12] N. Buduma, *Fundamentals of Deep Learning: Designing Next-Generation Artificial Intelligence Algorithms*. O’Reilly Media, 2017.
- [13] J. Patterson and A. Gibson, *Deep learning: A practitioner’s approach*. O’Reilly Media, 2017.
- [14] A. Karpathy, “Deep reinforcement learning: Pong from pixels.” [Online]. Available: <http://karpathy.github.io/2016/05/31/rl/>

- [15] T. Matiisen, “Demystifying deep reinforcement learning — computational neuroscience lab.” [Online]. Available: <http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/>
- [16] V. M. et al., “Playing atari with deep reinforcement learning.” [Online]. Available: <http://arxiv.org/pdf/1312.5602>
- [17] D. Silver, “Tutorial: Deep reinforcement learning.” [Online]. Available: http://hunch.net/~beygel/deep_rl_tutorial.pdf
- [18] F. S. Melo, “Convergence of q-learning: a simple proof.” [Online]. Available: <http://users.isr.ist.utl.pt/~mtjspaam/readingGroup/ProofQlearning.pdf>
- [19] V. M. et al., “Human-level control through deep reinforcement learning.” [Online]. Available: <https://www.nature.com/articles/nature14236.pdf>
- [20] M. Brenner, “Training a goal-oriented chatbot with deep reinforcement learning,” 2018, accessed November 2020. [Online]. Available: <https://towardsdatascience.com/training-a-goal-oriented-chatbot-with-deep-reinforcement-learning-part-i-introduction-and-dce3af21d383/>
- [21] T. Zhao and M. Eskenazin, “Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning,” *In the Annual SIGdial Meeting on Discourse and Dialogue (SIGDIAL)*, 2016.
- [22] A. Aubret, L. Matignon, and S. Hassas, “A survey on intrinsic motivation in reinforcement learning,” *ArXiv*, vol. abs/1908.06976, 2019.
- [23] A. Mehrabian and J. A. Russell, *An Approach to Environmental Psychology*. Cambridge: MIT Press, 1974.
- [24] R. Ryan and E. Deci, “Intrinsic and extrinsic motivations: Classic definitions and new directions,” *Contemporary Educational Psychology*, vol. 25, pp. 54–67, 01 2000.
- [25] J. Deriu, A. Rodrigo, A. Otegi, G. Echegoyen, S. Rosset, E. Agirre, and M. Cieliebak, “Survey on evaluation methods for dialogue systems,” *ArXiv*, vol. abs/1905.04071, 2020.