SPRING

GRANT AGREEMENT N. 871245

Deliverable D6.1

Neural network architecture specification and design

Due Date: 28/02/2021
Main Author: INRIA
Contributors: UNITN, CVUT, HWU, BIU
Dissemination: Public Deliverable

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 871245.

DOCUMENT FACTSHEET

Deliverable no.	D6.1: Neural network architecture specification and design
Responsible Partner	INRIA
Work Package	WP6: WP Name
Task	T6.1: Deep Architectures for Conversational System and Non-verbal Behaviour
Version & Date	VFinal, 24/02/2021
Dissemination level	[X] PU (public) [ ] CO (confidential)

CONTRIBUTORS AND HISTORY

Version	Editor	Date	Change Log
V1	INRIA	12/02/2021	First Draft
VFinal	INRIA, HWU	24/02/2021	Answered comments from reviewers + added contribution from HWU

APPROVALS

Authors/editors	INRIA + UNITN, CVUT, HWU, BIU
Task Leader	INRIA
WP Leader	INRIA

Contents

1 Deep reinforcement learning architecture for non-verbal behavior 3

1.1 Reinforcement Learning . . . 3

1.1.1 Reinforcement Learning Algorithms . . . 4

1.2 Deep Reinforcement Learning Framework . . . 7

1.3 Transfer Learning and Meta Reinforcement Learning . . . 7

1.4 Learning Environments and Task Definitions . . . 8

2 Model Predictive Control for low-level robot control 9

2.1 State-space models and Model Predictive Control . . . 9

2.2 Social constraints . . . 10

2.3 Implementation . . . 10

3 Neural architectures for the social interaction behavior 11

3.1 Representation of the Social State during Human-Robot Interaction . . . 11

3.2 Situated Social Decision Making . . . 12

4 Design of neural architectures for conversational systems 15

4.1 Deep Reinforcement Learning for multi-modal social situations . . . 16

4.2 Transformer models for open-domain and task-based conversation . . . 16

4.3 Transfer Learning for bootstrapping task-based systems . . . 16

4.4 Multimodal deep learning for vision and language grounding . . . 17

4.5 Neural methods for Natural Language Understanding (NLU) . . . 17

Contents of the Deliverable

The objective of Work Package 6 is to develop and implement methodologies enabling the robot to automatically learn action policies to decide

1. how to explore the environment
2. how to move towards one or several persons in order to improve the quality of the sensory data (images and acoustic signals)
3. how to attract the attention of the selected persons in order to facilitate face-to-face communication
4. action policies for multi-party conversation.

The different modules in this work package take as input the high level information provided by other work packages: a map of the environment (WP2), human behavior (WP4), localization and identification of people with respect to the robot (WP3). WP3 can also provide some metrics about audio-visual quality. The outputs are given as robot actions, in the form of gestures, conversation actions, as well as motor commands.

In this document, we present the different architectures that will be developed in the SPRING project to implement these behaviors. For both verbal and non-verbal behavior, we combine machine learning architectures with non-learned control schemes: obtaining labeled or even varied data for social robotic scenarios is hard. Therefore, the whole architecture of the robot is not completely centered around deep learning, but some parts use a-priori knowledge and ad-hoc methods. Learned and non learned methods are split in different sections, to emphasize the complementarity of both approaches.

The general architecture of the robot software will be composed of several modules: a social interaction planner, deciding who to interact with and how, a navigation module, and a conversation system. Eventually, all these components will interact with each other. All the modules will be integrated in the dedicated repository on the SPRING project organization on the INRIA Gitlab [1]. As per European Commission requirements, the repository will be available to the public for a duration of at least four years after the end of the SPRING project. They will use ROS [Stanford Artificial Intelligence Laboratory et al.] to communicate with each other and with the robot motor commands.

The document is organized as follows: in section 1, we present the deep learning architectures that will be developed for the non-verbal behavior of ARI; in section 2, we present the non-learned control scheme which will be combined with these architecture to provide the low-level controller of the robot. Section 3 presents the architectures for learning the social interaction behavior. Section 4 presents the architectures and method used for action policies in multi-party conversation.

[1] https://gitlab.inria.fr/spring/wp6_robot_behavior

1 Deep reinforcement learning architecture for non-verbal behavior

Non-verbal behaviors in the SPRING project are comprised of navigation and gesture (mainly arm, head and eye movements) behaviors. Different navigation and gesture tasks are part of the movement set that have to be provided for the ARI robot. For example, to navigate the robot to a goal position while avoiding obstacles, or to navigate the robot towards a person or a group of people to start a conversation. Many of these tasks are complex and based on high-dimensional variables such as the camera or auditory input from the robot. The classical approach to hand-engineer the behaviors for all these scenarios is too time-consuming and might not be possible in some cases due to the task complexity. As a solution, the SPRING project is using the deep reinforcement learning framework to learn most of the navigation and some of the gesture behaviors. Deep reinforcement learning is a machine learning approach to learn multi-step decision tasks while being able to handle complex and high-dimensional state input.

This section introduces first the general framework of reinforcement learning (Sec. 1.1) before discussing deep reinforcement learning (Sec. 1.2). Sec. 1.3 outlines the framework of transfer and meta learning. These mechanisms will be applied during the SPRING project to reduce the high amount of data and robotic experiments that are usually needed for deep reinforcement learning. The final section discusses the learning environment and how the reinforcement learning tasks are defined (Sec. 1.4).

1.1 Reinforcement Learning

This section introduces fundamental aspects of the reinforcement learning framework. Please refer to [Sutton and Barto, 2018] for a comprehensive introduction. Reinforcement learning uses the framework of Markov Decision Processes (MDP's) to formalize its learning tasks [Bellman, 1957, Puterman, 1994]. A MDP is a tuple:

$$\mathrm{MDP} = (S, A, T, R) \ ,$$

where S is a set of states, and A a set of actions. T is a transition probability function with $T : S \times A \times S \mapsto [0, 1]$, and R a reward function with $R : S \times A \times S \mapsto \mathbb{R}$.

States S are possible situations in which the agent has to make a decision. The agent in the SPRING project is the ARI robot. States can be discrete $S \in \mathbb{N}$, continuous ($S \in \mathbb{R}$), and multi-dimensional ($S \in \mathbb{R}^n$). In a state, the agent has to choose between a set of actions $A = \{a^1, \ldots, a^K\}$. Actions are usually a discrete finite set $A \in \mathbb{N}$ but can also be continuous and multi-dimensional ($A \in \mathbb{R}^n$). Depending on the agent's current state s_t at time point t and its action a_t, it transitions to the next state s_{t+1}. The transition function T defines a probability distribution over the successor states, given the current state and action: $T(s_t, a_t, s_{t+1}) = \Pr(s_{t+1}|s_t, a_t)$. Transitions fulfill the Markov Property, i.e. a transition only depends on the current state and not on previous states. The time of a transition is measured in discrete steps with $t = (0, 1, \ldots, T)$.

For each transition, the agent gains a reward according to the reward function R. The reward is a real-valued scalar and the general goal of the agent is to maximize the sum over the reward. The reward function can be defined over different elements of a transition at time point t. The reward function might depend only on the state and action at t: $r_t = R(s_t, a_t)$, or only on the state it transitions into: $r_t = R(s_{t+1})$, but it can also depend on all three components: $r_t = R(s_t, a_t, s_{t+1})$. Rewards may also be probabilistic. In this case, the reward function is the probability distribution over the rewards for a transition: $R(s_t, a_t, s_{t+1}) = \Pr(r_t|s_t, a_t, s_{t+1})$.

The agent starts to interact with a MDP by initializing in a start state $s_{t=0}$ at time point $t = 0$ according to an initial state distribution. From there, the agent decides upon an action $a_{t=0}$, transitions to the next state $s_{t=1}$, and receives a reward $r_{t=0}$. The behavior of an agent, i.e. which action it chooses, is defined by its policy π. A policy can be either a deterministic mapping from a state to an action: $a = \pi(s)$, or a probability distribution over the actions per state: $\pi(a; s) = \Pr(a|s)$.

Two major types of tasks are distinguished: episodic and continuous MDPs. Episodic MDPs consist of several episodes or runs. Each episode starts in an initial state and ends if a terminal condition is fulfilled. In many cases, a set $S^G \subset S$ of goals or terminal states are defined. An episode ends if the agent enters one of them. Continuous MDPs have no terminal condition. In that case, the number of steps goes to infinity. Control problems are often formulated as continuous tasks, for example, in the case of a pendulum swing-up task where an inverted pendulum should the balanced at its top position. Episodic tasks are often used for decision problems, for example, to find the optimal path to a goal state.

The goal or objective of an agent is to find a policy that maximizes the rewards that the agent can gather in a MDP. Two major types of objectives exist. The first is the maximization of the expected discounted future reward sum:

$$\mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \;=\; \mathrm{E}[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots]\ , \tag{1}$$

where $\gamma \in [0, 1)$ is a discount factor. The objective is formulated as an expectation because trajectories are stochastic. Trajectories are a random variable because the transition or the reward function can be probabilistic, or the policy itself may be probabilistic. The sum of the expected discounted future reward is mainly used as objective for episodic tasks. The effect of the discount factor γ is to reduce the weight of temporally distant rewards in the sum. The second common objective is the maximization of the expected average reward:

$$\lim_{T\to\infty} \mathrm{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} r_t\right]\ . \tag{2}$$

This objective is mainly used in the case of continuous tasks, because it has a stronger bound on the sum as t goes to infinity. Which objective is used depends on the type of task. Moreover, different algorithms exist for maximizing each objective.

1.1.1 Reinforcement Learning Algorithms

The following section gives a general overview over reinforcement learning algorithms and their classification. The goal of all algorithms is to learn the policy that maximizes the reward sum in a MDP, as defined by the objective (Eqs. 1 or 2). Algorithms can be categorized according to the information they use to optimize the policy and how they represent the policy.

One major classification is the distinction between model-free and model-based approaches. Model-free approaches learn the policy based on data from observed trajectories of the agent interacting with the MDP. Model-based methods use a model of the MDP, where the model describes the transition probability and the reward function. The model can be either given, or learned from observed trajectories.

Moreover, algorithms are differentiated into value-based, policy-based and actor-critic approaches. Value-based algorithms learn a value function to optimize the expected reward sum. The value for a state represents the expected future reward sum if the agent follows a certain policy starting from that state. This information is used to find the policy that maximizes the value. In contrast, policy-based algorithms do not learn a value function to optimize the policy. Instead the policy is optimized directly. Actor-critic methods represent a mixture of value-based and policy-based methods as they learn a value-function and optimize the policy in parallel.

For the SPRING project model-free reinforcement learning methods will be mainly utilized. Their advantage over model-based methods is that they do not need a model of the environment. Defining or learning a model in the context of social robotics is especially problematic as humans with which the robot needs to interact show rich and often difficult to model behavior. This makes it difficult to apply model-based approaches.

The following sections will give an introduction to the three major classes of model-free algorithms: value-based, policy-based and actor-critic methods.

Value-Based Algorithms

The central element of value-based algorithms is the value function [Bellman, 1957] which associates a value with states or state-action pairs. Values describe predictions about the future outcome of being in these states or taking these actions in terms of the expected future reward sum. Based on this notion, an agent can decide to go to states and to take actions that maximize the reward sum. We concentrate here on the objective of maximizing the discounted future reward sum (Eq. 1).

The value function can be formulated as a state-value function $V(s)$ or an action-value function $Q(s, a)$, called a Q-function. The value of a state is the expected discounted reward sum starting at state s and using policy π in successive states:

$$V^{\pi}(s_t) \;\equiv\; E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right]\ ,$$

where $\gamma \in [0, 1)$ is the discount factor. The value of a state-action pair, called a Q-value, is the expected discounted reward sum starting from s using action a and following policy π:

$$Q^\pi(s_t, a_t) \equiv E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right] .$$

Bellman [1957] showed that the value function can be recursively defined in terms of the immediate reward and the value of the succeeding state:

$$\begin{aligned} V^\pi(s_t) &= \mathrm{E}_\pi\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots\right] \\ &= \mathrm{E}_\pi\left[r_t + \gamma V^\pi(s_{t+1})\right] \\ &= \sum_{s_{t+1}\in S} T(s_t, a_t, s_{t+1})\left(\sum_{a_t \in A} \pi(a_t; s_t) R(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1})\right) , \end{aligned}$$

This so called Bellman Equation can be also formulated for the Q-function:

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathrm{E}_\pi\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots\right] \\ &= \mathrm{E}_\pi\left[r_t + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1}))\right] \\ &= \sum_{s_{t+1}\in S} T(s_t, a_t, s_{t+1})\left(R(s_t, a_t, s_{t+1}) + \gamma \sum_{a_{t+1}\in A} \pi(a_{t+1}; s_{t+1}) Q^\pi(s_{t+1}, a_{t+1})\right). \end{aligned}$$

Values depend on a policy and measure the expected discounted future reward that the policy produces. The goal is to find the optimal policy π^* that maximizes the value function for each state ($\forall s \in S, \pi : \; V^{\pi^*}(s) \geq V^\pi(s)$). Bellman showed that the optimal value can be reached, if for each state, the action that maximizes the immediate reward and the optimal discounted value of the successor state is chosen:

$$V^*(s_t) = \max_{a_t} \sum_{s_{t+1}\in S} T(s_t, a_t, s_{t+1})\Big(R(s_t, a_t) + \gamma V^*(s_{t+1})\Big) ,$$

$$Q^*(s_t, a_t) = \sum_{s_{t+1}\in S} T(s_t, a_t, s_{t+1})\Big(R(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})\Big) .$$

If the optimal state-value function is given, the optimal deterministic policy can be deduced by:

$$\pi^*(s_t) = \operatorname*{argmax}_{a_t} \sum_{s_{t+1}\in S} T(s_t, a_t, s_{t+1})\Big(R(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\Big) .$$

The expression shows that the transition probability function T, and therefore a model is still necessary to identify the optimal policy. This is not necessary for the optimal Q-function where the optimal policy is given by:

$$\pi^*(s_t) = \operatorname*{argmax}_{a_t} Q^*(s_t, a_t) .$$

As a consequence, model-free methods generally use Q-functions instead of state-value functions.

In summary, the value function represents the expected future discounted reward sum that an agent will receive if it follows a certain policy. Given the optimal value function the optimal policy for an MDP can be deduced. Value-based methods use this property by learning the optimal value-function to find the optimal policy.

One of the main approaches to learn the value function is temporal difference (TD) learning [Sutton and Barto, 2018]. These methods make use of the recursive property of the value function, allowing them to perform bootstrapping. TD learning predicts the Q-function for an agent with policy π. It starts with an initial Q-function that can be arbitrarily initialized. The Q-function is updated after each transition of the agent from state s_t with action a_t to state s_{t+1}. $Q^\pi(s_t, a_t)$ is the agent's prediction of the sum of the expected reward $\mathrm{E}[r_t]$ of this step and the expected discounted sum over all future steps according to policy π: $\mathrm{E}\big[\gamma Q^\pi(s_{t+1}, a_{t+1})\big]$. After the transition the agent observed the reward r_t and the Q-value of the next state. Based on this observation the TD error δ_t is computed:

$$\delta_t = \underbrace{r_t + \gamma Q^\pi(s_{t+1}, a_{t+1})}_{\text{observation}} - \underbrace{Q^\pi(s_t, a_t)}_{\text{prediction}} , \tag{3}$$

which is the difference between the prediction and the observation. The next action a_{t+1} follows the policy of the agent. The difference between the prediction and observation is employed to update the prediction using the Robbins-Monro approach for stochastic approximation [Robbins and Monro, 1951]:

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha \delta_t \ ,$$

where the learning rate $\alpha \in [0, 1]$ defines how strong the update is. If α is not constant, but decays over time in an appropriate way, the algorithm will converge to the true prediction of Q^π.

A prominent TD algorithm is Q-learning [Watkins, 1989, Watkins and Dayan, 1992]. Its purpose is to learn the values for the optimal policy π^*. It achieves this by updating the prediction after an observation with the Q-value for the action that maximizes the value over the next state:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) \ . \tag{4}$$

Because of the max operator, Q-learning is an off-policy algorithm. It learns the value function of the optimal policy regardless of the policy that is used to produce the observations.

Policy-Based Methods

Policy-based methods represent a different class of reinforcement learning algorithms. They learn directly the policy without learning or representing a value-function. For this purpose, the policy $\pi(a; s, \theta)$ is usually parameterized by a vector of parameters $\theta \in \mathbb{R}^n$. The parameters determine the probabilities over the actions for each state. Often, the average reward objective is used in this context (Eq. 2) and the goal is to find the parameters that maximize it:

$$\operatorname*{argmax}_\theta \mathrm{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \ .$$

Many methods in this class use a gradient ascent approach by computing the gradient of the policy parameters with respect to the objective [Deisenroth et al., 2011]:

$$\nabla J(\theta) = \nabla \mathrm{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \ .$$

The gradient determines how the parameters should be changed to improve the objective. It can be computed for a point in the parameter space by sampling trajectories with policies based on parameters around it. Parameters are then adapted according to the gradient and the process is repeated:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta) \ .$$

Actor-Critic Methods

Actor-critic algorithms are a mix of value-based and policy-based methods. They learn a policy, called actor, and a value function, called critic, together. In difference to value-based methods, the critic is usually a state-value function instead of a Q-function. The critic is learned via the TD learning mechanism, i.e. for a transition the value is updated via:

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t \quad \text{with} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \ .$$

The parameters of the actor π_θ are then updated according to the TD error δ_t:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \pi_\theta(a_t | s_t) \delta_t \ ,$$

so that if the used action a_t results in a higher expected return than predicted by the current critic $V(s_t)$, its probability of being chosen increases. If it results in a lower expected return as predicted, its probability gets reduced.

An advantage of actor-critic methods over value-based methods is that they can handle continuous action spaces. These are difficult to learn with Q-functions, as the max-operator is needed over all actions to identify the best action which is infeasible to compute for an infinite continuous action set. Moreover, the state-value function is less complex as the Q-function. An advantage over policy based methods is that the usage of the value function reduces the variance of the policy updates and accelerates learning [Sutton and Barto, 2018].

1.2 Deep Reinforcement Learning Framework

An important part of reinforcement learning is how its components such as the value function or policy are represented. This representation depends on the type of state spaces of the problems. For problems with finite and discrete state spaces a simple tabular representation can be used, i.e. each value is represented by a scalar value in the memory. Nonetheless, in the SPRING project, the state space is continuous and often high-dimensional $S \in \mathbb{R}^n$, consisting of multi-modal continuous sensor information or preprocessed features such as camera images, a semantic 2D map, auditory input, or human behavior descriptions. Function approximators are needed to represent a value-function (V or Q) or a policy (π) over such a continuous space. The approximators represent continuous functions using a parameterized models, for example to represent the Q-function:

$$Q^{\pi}_{\theta}(s_t, a_t) = \mathrm{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k * r_{t+k}\right] \ ,$$

where $\theta \in \mathbb{R}^N$ are the parameters of the model that define the approximated Q-function. The goal of a TD algorithms is to learn the parameters that optimize the Q-function:

$$\theta^* = \underset{\theta}{\mathrm{argmax}}\, Q_\theta \ .$$

Early function approximators such as linear functions, decision trees, or tile coding [Sutton and Barto, 2018] represent "shallow" approximators as their complexity and their number of parameters are low. Such models are only appropriate for problems with low-dimensional state spaces. The framework of deep reinforcement learning (DRL) uses instead deep neural networks (DNNs) having many network layers and parameters to represent its components (See Li [2017] for a review about DRL). DNNs are able to represent high-dimensional functions and the recent breakthrough in deep learning allowed to efficiently train such networks [Goodfellow et al., 2016].

A problem with the usage of non-linear function approximators such as deep neural networks is that classical RL methods become unstable or even divergent [Tsitsiklis and Van Roy, 1997]. Recent advances in DRL have overcome this problems by introducing techniques such as replay buffers or target networks. Several algorithms have been proposed to allow the usage of deep neural networks together with reinforcement learning such as value-based methods: Deep Q Networks [Mnih et al., 2015], policy based-methods: Guided Policy Search [Levine and Koltun, 2013, Levine et al., 2016], and actor-critic methods: Soft Actor Critic [Haarnoja et al., 2018]. The SPRING project will use such algorithms as its basic framework to learn the robotic behaviors.

For the implementation and training of the DNN structures the PyTorch framework [Paszke et al., 2017] will be used. Its relevant features for the SPRING project are:

- easy implementation of deep neural network models,
- support of automatic gradient computation needed for the training of the networks,
- the usage of graphics processing units (GPU) for efficient computation,
- a vast library of existing DNN and DRL structures and algorithms based on PyTorch,
- open source.

1.3 Transfer Learning and Meta Reinforcement Learning

A problematic point of deep reinforcement learning is the amount of data that is required to learn appropriate policies. The agent needs to explore a lot of the state space and observe the outcomes of different actions to identify the best action per state. In the context of robotics, this learning process takes a long time.

Furthermore, the learned behavior depends on the reward function that has to be defined by the user. However, it is often not foreseeable what behavior will result from a reward function. For example, in a navigation task where the robot has to approach a human, the reward function could have a component which punishes strong movements. This should ensures that the robot is not learning a policy that behaves erratic or approaches a human with a too high velocity. How strong this component influences the whole reward function can lead to vastly different behaviors. If the punishment is too small, then the robot might approach a human too fast and is perceived as threatening. If the punishment is too large, then the robot might approach too slow or does not move around obstacles to reach the human. Often, the reward function needs to be adapted to learn an appropriate behavior. For classical reinforcement learning the task would have to be learned from scratch for each new reward function costing a lot of time.

As a solution to these problems the SPRING project will utilize transfer learning and meta learning techniques. Transfer and meta learning methods transfer knowledge from already learned source tasks to target tasks [Taylor and Stone, 2009, Lazaric, 2012, Zhu et al., 2020]. The goal is to improve learning performance on the target task. This enhancement can improve the learning speed, resulting in a higher initial performance, or a higher final performance after learning on the target task. Methods for two scenarios will be explored and applied. The first is the ”simulation to real transfer” (sim2real) and the second the transfer of knowledge between tasks with changing reward functions.

The first approach targets to reduce the learning time on the physical robot using initially a simulation of the robot and its environment to train the robot behavior. Training in a simulation reduces the need of costly experiments with the physical robot. Nonetheless, a simulation can not replicate exactly the real physical world and the relevant task features such as the reaction of humans. The policies learned in simulation perform usually not well on the physical robot directly. As a consequence, various methods have been explored to allow the transfer of policies learned in simulation to the physical system. For example, by learning a policy that has to cope with different simulation behaviors to become more robust to different environments [Chebotar et al., 2019].

The second approach targets to reduce the cost of learning for tasks where the reward function is changing such as due to incremental adaptations of it by the user. The goal is to transfer knowledge from already trained tasks under different reward functions to the same task but with a new reward function. Existing methods have for example achieved such transfers by reevaluating the performance of previously learned policies from tasks with different reward functions in a new task [Barreto et al., 2016]. Given a new reward function, the best policy that has been learned so far can be selected and used as an initial policy for the new task. Such mechanisms reduce the need for exploration in the new task and speed up the learning process. During the SPRING project such methods and sim2real transfer approaches will be used and further developed.

1.4 Learning Environments and Task Definitions

This section describes the general environment used for the reinforcement learning experiments and how the different components of the tasks are defined, i.e. the state space, action space, and the reward functions.

The SPRING project will use simulations and experiments on the physical robot to learn the robot behaviors. Different simulators are utilized depending on the task requirements. For example, a 2D simulator will be utilized to learn approaching behaviors of the robot to individuals and groups, simulating the 2D positions, body and head orientations and velocities of people in hospital scenarios. The targeted behavior is initially learned using simulator. A sim2real mechanism is then used to transfer the learned policy to the robot and finish the training using real world scenarios (Sec. 1.3).

The state space S for the reinforcement learning tasks will depend on the task and can range from low-level sensor input, such as camera images or audio recordings, to high-level features such as the positions and postures of people. Deep neural networks are used to learn the appropriate representations from these state variables to represent successfully value-functions or policies (Sec. 1.2).

The action space A will be comprised of high-level actions that are then executed by the low-level controller (Sec. 2.1). For example, in the case of navigating the robot for joining a group of people, actions represent points in the space around the robot which can be several tenth of cm away from the robot. The low-level controller is then responsible to navigate the robot to the selected point.

The reward function R for a task will be task dependent defined. For the SPRING project different ways to define the reward function will be used. Reward functions will be composed of different components $r_i \in \mathbb{R}$ that are computed based on task features $\phi : S \mapsto R^D$. Components are then combined, for example using a weighted linear combination:

$$R(s) = \sum_i w_i r_i(\phi(s)) \ ,$$

where w_i are weights for the different components. In many cases the reward components and the associated features are hand-defined. For example, in a navigation task to approach a person one reward component $r_d(d) = (d - d^*)^2$ provides negative reward if the robot has not yet reached the optimal distance to the target person d^*, using as feature the distance to this person $\phi(s) = d$. Besides hand defined reward components, SPRING will also include reward components from existing models about naturalistic behaviors, for example from social force models about the approach of humans [Gao et al., 2019]. Furthermore, data about the behavior of people in similar situations to the robot will be collected. This data will be used to evaluate how similar the robot behaves in the same situation which is given as a reward signal so that the robot is inclined to learn similar behaviors.

2 Model Predictive Control for low-level robot control

CNNs have improved state-of-the art in many pattern recognition tasks, especially in vision. In the past years, it has been applied successfully to more complex tasks than simple pattern recognition: for example, end-to-end learning of self-driving cars [Bojarski et al., 2016]. This raises the question of applying such end-to-end frameworks to our goals. In SPRING, we decide to use a hybrid architecture which combines machine learning architectures with ad-hoc non-learned constraints and goals. The reasons for this are multiple fold: first, compared to a self-driving car scenario, in a social robotics scenario the targets are not so well defined: social interactions are often implicit... End-to-end learning also needs huge amounts of both computing power and training data that we might not be able to provide in the SPRING project. Furthermore, data on social interactions is hard to acquire, both for ethical and for practical reasons. Therefore, we decide to use tradition control methods for the robot motors, and to incorporate a priori knowledge about social conventions in the architecture of the robot.

2.1 State-space models and Model Predictive Control

Model Predictive Control (MPC) [Camacho and Alba, 2013] is a well established method for controlling various systems for which a state-space model can be formulated. This generic control scheme can be adapted to incorporate learned features or penalties, as well as physical on the constraints. We will now describe the general architecture for the MPC controller of the robot. In it's most generic form, an MPC controller assumes that a state-space model is provided. In the following, we will consider a discrete setting, but the framework can be expressed in a continuous setting very similarly. Let $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ denote respectively the state and control at time t. A state-space model provides a function f, such that:

$$\boldsymbol{x}(t+1) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)). \tag{5}$$

We also consider constraints on the system, defined as a function g:

$$g(\boldsymbol{x}(t)) \geq 0, \forall t \in \mathbb{N}. \tag{6}$$

Let us consider a target state $\bar{\boldsymbol{x}}$, a loss function l and a regularization function λ. Given a finite horizon N, J is the following functional:

$$J_N(\boldsymbol{x}(0), \boldsymbol{u}(0), \cdots, \boldsymbol{u}(N-1)) = \sum_{t=1}^{N} l(\boldsymbol{x}(t), \hat{\boldsymbol{x}}) + \lambda(\boldsymbol{u}(t)).$$

MPC consists in repeatedly applying the following scheme:

1. Estimate current state $\boldsymbol{x}(0)$;
2. Minimize J_N with respect to controls $\boldsymbol{u}(0), \cdots, \boldsymbol{u}(N-1)$, subject to (5, 6);
3. Apply first control $\boldsymbol{u}$ until $t = 1$;
4. Go back to 1;

The robot state is described by its joint angles (α_1, α_2), its position (x, y) and its orientation θ:

$$\boldsymbol{x}(t) = (\alpha_1(t), \alpha_2(t), \theta(t), x(t), y(t)).$$

The robot control is done in the ego frame, i.e. $x(0) = y(0) = \theta(0) = 0$. We assume that the motors are velocity controlled, i.e. each motor takes a target velocity as input. A simple change of coordinate changes wheel velocities to a linear velocity v and an angular velocity ω:

$$\boldsymbol{u}(t) = (\dot{\alpha_1}(t), \dot{\alpha_2}(t), \omega, v).$$

Then the forward model is given by, given a discretization timestep T:

$$\begin{cases} x(t+1) & = x(t) - T\sin(\theta(t))v \\ y(t+1) & = y(t) + T\cos(\theta(t))v \\ \theta(t+1) & = \theta(t) + T\omega \\ \alpha_1(t+1) & = \alpha_1(t) + T\dot{\alpha_1}(t) \\ \alpha_2(t+1) & = \alpha_2(t) + T\dot{\alpha_2}(t) \end{cases}$$

The model devised for the robotic base control is therefore a simple non-linear system. Due to its nonlinearity, linear MPC control schemes cannot be used, directly, and we will need to be careful about performance issues when using the optimization scheme described above. Nonetheless, with carefully computed derivatives, early experiments show that the controller should be able to run in real time on the robot.

2.2 Social constraints

When people engage in conversation with one another, they organize themselves in distinctive spatial arrangement. The knowledge about these formations, called F-formations, can be utilized by the robot. For example, a group organized in a F-formation will share a private space, called the o-space, reserved for the group. This space should not be occupied by a social robot [Kendon, 2010]. Another example of social constraints would be to respect privacy spaces of people the robot is interacting with depending on the level of interaction.

These constraints should be integrated in the control of the robot, for example in the form of a velocity map, as in the works of Truong and Ngo [2017]. This velocity map, expressed as a function v of the state of the robot, can be integrated in the loss function described above:

$$l(\boldsymbol{x}(t), \hat{\boldsymbol{x}}) = v(\boldsymbol{x}) + a|\boldsymbol{x}(t) - \hat{\boldsymbol{x}}|.$$

We will also integrate rewards computed from reinforcement learning techniques (see 1.4) into the loss function of the MPC controller.

2.3 Implementation

Implementation is done using the Jax [Bradbury et al., 2018] Python library. This allows all python functions to be JIT-compiled, and enables automatic differentiation. This is crucial to get real time performance for the MPC controller.

Goals for the MPC controller will be determined by the different Reinforcement Learning algorithms described in section 1.

3 Neural architectures for the social interaction behavior

The Socially Assistive Robot in the SPRING project needs to be able to understand various individual and group situations and take appropriate decisions, e.g. identify persons that need assistance (such as patients that have been waiting for a long time and who might be anxious), engage in face-to-face multi-modal dialogue with a patient, a family member, a staff member, or with a party of them, or accompany a patient across the hospital. The architectures developed through this section will help empower the SPRING ARI robot with the skills needed for situated interactions, i.e. interactions grounded upon the social, semantic, behavioral, and geometric representation of the immediate environment.

A social interaction behavior system as outlined here requires components such as a social decision maker, a task planner, and a multi-user conversational manager, to enable situated interactions with multiple users at the same time. To allow these components to interact appropriately, a tight integration of the different input modalities will be required, using a representation of the social state, e.g., mapping and localization, visual/audio features, human behavior understanding.

3.1 Representation of the Social State during Human-Robot Interaction

The SPRING system will feature different network streams from various modules, e.g., semantic mapping and localization (WP2), visual/audio features (WP3), human behavior understanding (WP4), robot proprioception/actions (WP6). All of this data must be combined in a manner which allows the decision-making systems to select appropriate social behaviors. In order to create robots able to move, see, hear and communicate in a social context with multiple agents, and properly fulfil social roles and successfully execute social tasks, we need to model the human-robot interactions into a representation of the *Social State*.

The *Social State* representation is tasked with turning the continuous stream of messages produced by the low-level input and output components into a discrete representation of the world, the robot, and all entities in the scene, integrating social, interaction-based, and task-based properties. This representation will exploit the the structured cues from WP2, WP3 and WP4 to describe multi-party interactions, devise social interaction plans, and support the robot decision making for the high level planner and the conversational manager for maximizing the robot's execution strategies for social interaction and communication.

Developing SARs with the capacity of performing natural and continuous interactions in a social context with multiple agents in an 'open-domain' requires the robot to show the ability to track and ascribe social meaning to its sensory information. It must explore the environment and understand what the environment affords, including which objects, actions, events, and scene information can be extracted from the sensory data. It must also 'know' when to perform communicative actions and decide whom to respond to, and when and how to address one or more people. It must track the state of each agent, and track their conversations, determining what they are saying and to whom, where their attention is at, as well as predicting their goals, and their affective and emotional states, etc., and on the relationship between such elements [Rato et al., 2020].

We propose to organize the representations into four domains [García et al., 2020]: *1)* the external domain, with the representation of the scene; *2)* the behavior domain, with the representation of the person's actions/behaviors; *3)* the mental domain, with the representation of the person's attributed mental states; *4)* the dialogue domain, with the representation of the conversations.

The external domain models the social scenario in a representation of the scene, e.g. where is the interaction taking place? Who is taking part in the interaction? It models objects, places, structures, and agents and their relations in a way that is physically grounded from the environment by the robot's sensory information. The behavior domain models the behavior of the users and is a representation of the people interacting in the scene, which combines persistent data of the user for identification[2], i.e., name, id, role, and dynamic data of the user behavior, i.e., current activity, location, status. The mental domain which models the mental state attributions of the users is a representation of the internal state of the users – it tracks the intentions and beliefs of participants of the interaction, as well as predictions of the goals, motivations and emotional states, etc. The model of the dialogue state is a representation of the conversation, tracking what has been said and by whom, the intents expressed, the entities mentioned, and the topics of conversation during interactions among multiple users and the robot.

For the initial version of the *Social State* (see Table 1) the state is represented as a list of properties, from the four previously mentioned domains plus the internal robot state.

[2]Assuming consent of the user to do so.

Table 1: Social Scene Representation.

Feature	**Signal**	**Domain**	**Description**
locale	localization	Scene	Where the interaction is taking place.
agents	detection & tracking	Scene	List of agents in the scene.
objects	semantic segmentation	Scene	List of (salient) objects in the interaction.
rooms	semantic mapping	Scene	List of places.
scene	scene-graph	Scene	Graph/map of the scene.
ID	identification	behavior	ID of the person.
groupID	identification	behavior	Group they belong to.
role	behavior analysis	behavior	People role depending on interaction context.
activity	behavior analysis	behavior	Track current activity.
attention	gaze tracking	behavior	Track focus of attention.
location	localization	behavior	Track current location.
purpose	intention recognition	Mental	Agent's main goal.
currentTarget	intention recognition	Mental	Goal pursued (by the agent) at present.
emotionState	behavior analysis	Mental	Estimation of agent affective expression.
motivation	behavior analysis	Mental	Condition of the agent, i.e, engaged, waiting, etc.
speakerID	identification	Dialogue	ID of the speaker.
listenerIDs	identification	Dialogue	IDs of the listeners.
intent	NLU	Dialogue	The intent of the speaker.
entities	NLU	Dialogue	Entities in the dialogue.
topic	NLU	Dialogue	The current topic/task.
onset	ASR	Dialogue	When the dialogue starts.
transcript	ASR	Dialogue	Transcript of the dialogue.
robotLocation	localization	Internal	Robot current location.
robotAttention	robot gaze	Internal	Robot current focus of attention.
lastAction	robot behavior	Internal	Action executed by the robot.
robotGoal	robot planner	Internal	Goal pursued (by the robot) at present.
interactTarget	robot planner	Internal	Person(s) robot is interacting with.

3.2 Situated Social Decision Making

Managing situated interactions, as in the use case of SPRING, requires systems that can reason about their surroundings and interact in a natural manner in open-world, physically situated settings of human interactions, leveraging both the dialogue and social-physical context when making decisions [Bohus and Horvitz, 2019]. A first problem that must be dealt with is opening a communication channel. Situated interaction systems generally need to reason about and manage engagement, i.e., the process by which we initiate, maintain and terminate conversations, deciding who they are interacting with, and when [Bohus and Horvitz, 2009]

To equip social robots with the ability to detect a person who is willing to interact, in the initial stage a basic decision making system will be adopted from Romeo et al. [2019], that is able to decide if a person, in the field of view of the robot, is looking to initiate a conversation with the robot. This has been integrated with the the initial task scheduler and the initial conversational system of WP5 [Part et al., 2021] to create a first closed loop system for high-level robot control, able to use visual data to select a person to talk to and start a conversation.

The engagement decision-making network in Romeo et al. [2019] (see Figure 1), comprises 9 layers in total, excluding the input layer. The first convolutional layer takes as input grayscale images from the robot cameras, resized to 198 $\times$ 198 pixels, and applies 16 filters 9 $\times$ 9 with stride 3. The second and the third layer apply 32 and 64 filters 5 $\times$ 5 respectively, with stride 1. Each convolutional layer is followed by a rectifier linear unit (ReLU) function and a max pooling layer with pool size 2 $\times$ 2 and stride 2. The last convolutional-max pooling layer is followed by a fully-connected layer of 256 units. The final output layer is a fully-connected linear layer of 3 units, giving as output the probabilities for each of the 3 available actions to be the right one to be chosen in the current situation. The choice of the network is the action associated with the label having the highest probability for the input image.

The available actions are: (i) *wait*, not doing anything, for the cases in which no-one was detected as interested in engaging; (ii) *start* the interaction, addressing the person identified as willing to start the interaction with a direct

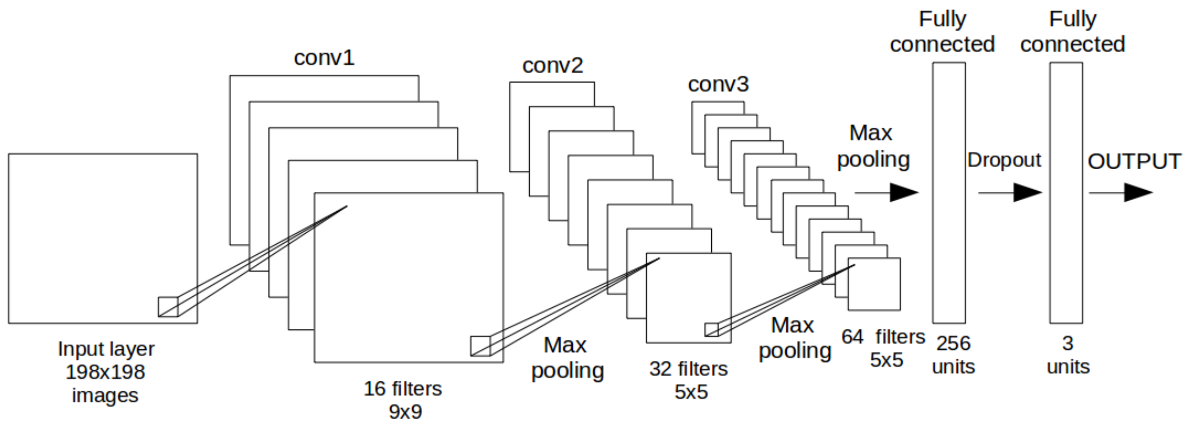

Figure 1: Architecture of the CNN (from Romeo et al. [2019]): 3 convolutional layers, each followed by a corresponding max pooling layer, and 2 fully-connected layers. The first convolutional layer applies 16 filters 9x9 with stride 3. The second and third apply 32 and 64 5x5 filters respectively, with stride 1. The last convolutional layer is followed by a dense layer of 256 units, after which a dropout layer is added. After applying the dropout, the output layer is a fully-connected linear layer of 3 units.

question, e.g "Do you like music?" or "Let's talk about sport, do you prefer football or basketball?", etc.; (iii) *call for* attention, a neutral action to take, when undecided to *start* the interaction, the robot will make a small gesture, e.g. 'move close', 'small wave', 'head nod', etc., to test whether the user is interested to engage. The start of an interaction decision-making process follows these steps: look at the scene happening in front of the robot; extract grayscale frames, using the camera on the forehead of the robot, and given as input to the CNN; from the output of the CNN, classifying the last 15 frames with respect to the actions that the robot could take (*wait*, *start* the interaction and *call for* attention), decide the action to perform, chosen as the higher average score from the last 15 frames.

The architecture in Romeo et al. [2019] uses nonverbal bodily social signals to determine engagement in human-human interactions, with a CNN in its decision-making process. Further work can explore broadening the decision-making capabilities, of the interaction, and extend it for multiple user, with a Reinforcement Learning (RL) component. Since interaction is a key component in both RL and social robotics, formulating social interactions with physically embodied social robots as sequential decision-making tasks can be a suitable approach which is gaining increasing attention in the robotics community.

Q-learning is the most commonly used RL method in social robotics. In Papaioannou et al. [2017a] an agent was trained using the standard Q-learning algorithm with simulated users and tested with the Pepper robot to assist visitors of a shopping mall by providing information about and directions to the shops, and current discounts in the shops among other things. It combines task-based and chat-style dialogue modalities in a scalable approach, allowing to easily enrich the used feature vector by including information from the robot's sensors in addition to verbal information. Weber et al. [2018] incorporated social signals in the learning process, using a two-dimensional vector containing probabilities of laughs and smiles for state representation, for a robot to adapt its sense of humor by using Q-learning with a linear function approximator.

Social interactions require features from high dimensional signals, as discussed in 3.1. We will explore Deep Reinforcement Learning approaches for scaling Reinforcement Learning problems with high-dimensional state spaces. Qureshi et al. [2016] utilizes deep Q-learning to enable a robot to learn social interaction skills from experience interacting with people in more complex scenarios where the robot can be approached by a group of people who are either willing or not willing to interact with it. They proposed a Multimodal Deep Q-Network (MDQN) to enable a robot to learn human-like interaction skills through a trial and error method with aim of enabling a robot with the ability to learn how to greet people. Their proposed multimodal DQN consists of two identical streams of CNN for action-value function estimation, one for grayscale frames and another for depth frames. The work in Qureshi et al. [2017] applied a variation of DQN, the Multimodal Deep Attention Recurrent Q-Network (MDARQN), to the same scenario. Here a recurrent attention model was used, which enabled the Q-network to focus on certain parts of the input image, (see Figure 2). Two identical Q-networks were used (one for grayscale frames and one for depth frames). Each Q-network consisted of convnets, a Long Short-term Memory (LSTM) network, and an attention network. Each of the streams of the MDARQN model were trained by using the back-propagation method. The outputs from the two streams were normalized separately and averaged to create output Q-values of MDARQN. Another study using a social robot and DQN was presented in Cuayáhuitl [2020] for efficiently training a humanoid robot to carry out joint multimodal activities together with humans that involve speaking, listening, gesturing, and learning. They used a CNN for recognizing game moves, i.e., hand-writing on the grid. These visual perceptions and the verbal conversations of the participant were given as an input to their modified DQN. The proposed DQN algorithm refines the action set at each step to allow the agent to learn to infer the effects of its actions (such as selecting the actions that lead to winning or to avoid losing). The reward consisted in predefined numerical values based on the performance of the robot in the game.

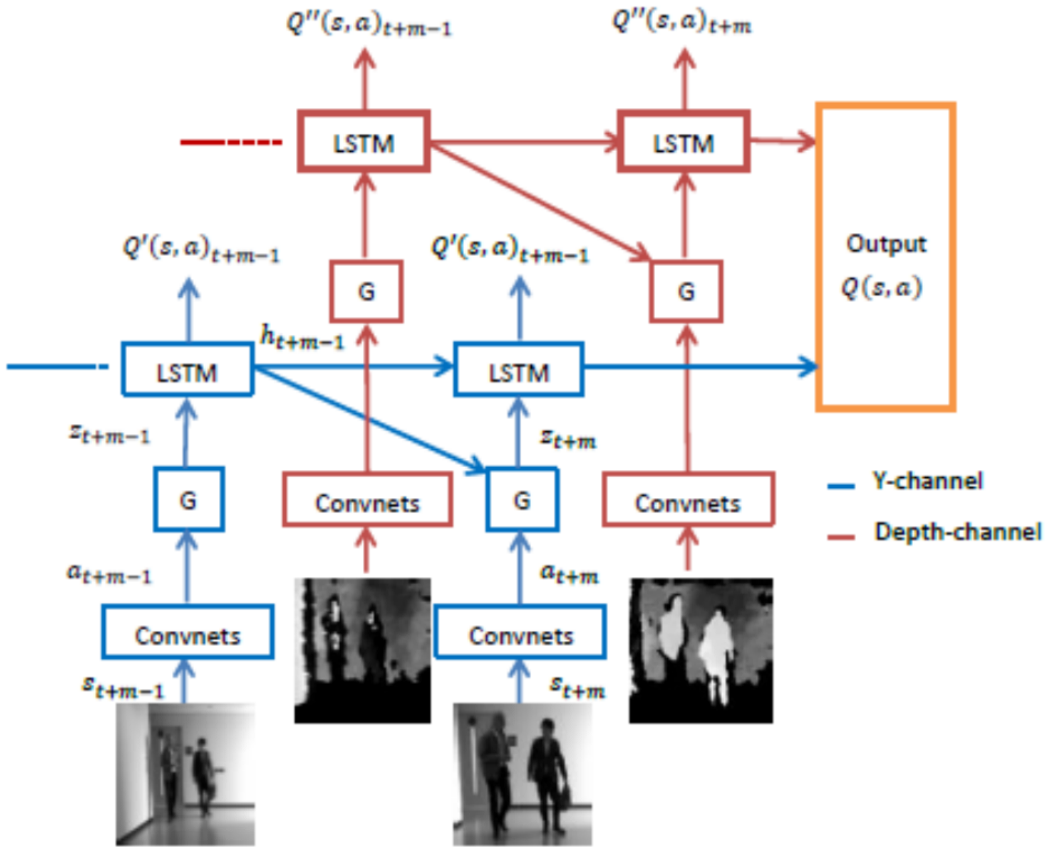

Figure 2: Multimodal Deep Attention Recurrent Q-Network (from Qureshi et al. [2017]): two Q-networks streams, each stream consists of three neural models. 1) Convnets; 2) Long Short-term Memory (LSTM) network; and 3) Attention network (G). The output Q-values from each Q-network stream are fused together for taking a greedy action.

We will explore data-driven machine learning approaches for learning social human-robot interaction strategies, and integrate the different SPRING input streams.

Keizer et al. [2013] uses human-human and human-robot data that was manually labeled to learn low-level sub-modules for how a bartender robot should interact with multiple customers, e.g., classifying user engagement, or saying pre-defined utterances.

The approach in Nanavati et al. [2020] autonomously learns interaction logic in a one-to-many shopkeeper-customer scenario. The proposed system utilizes neural networks (see Figure 3) to first learn, through an attention network, which customer actions are important to respond to and then learn via an Interaction network how the shopkeeper should respond to those important customer actions. The data-driven pipeline takes in low-level data of a human shopkeeper interacting with multiple customers and extracts high-level actions from that data, by first learning who to pay attention to (Attention Network), and then learning how to respond to that customer (Interaction Network). To train the Attention Network, they used labels from a Causal Inference Network that learned which customer most likely caused a shopkeeper action.

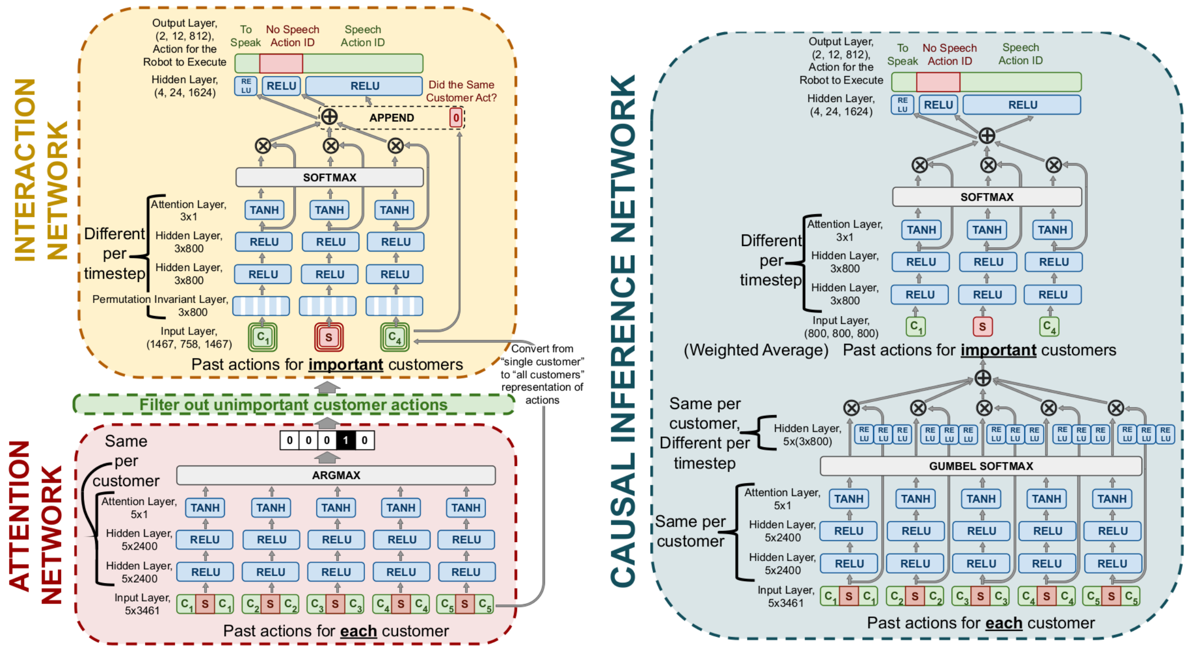
INTERACTION NETWORK
ATTENTION NETWORK
CAUSAL INFERENCE NETWORK
Output Layer, (2, 12, 812), Action for the Robot to Execute
To Speak
No Speech Action ID
Speech Action ID
Hidden Layer, (4, 24, 1624)
RELU
APPEND
Did the Same Customer Act?
SOFTMAX
Attention Layer, 3x1
Hidden Layer, 3x800
Permutation Invariant Layer, 3x800
Input Layer, (1467, 758, 1467)
Different per timestep
TANH
Past actions for important customers
Filter out unimportant customer actions
Convert from "single customer" to "all customers" representation of actions
Same per customer
ARGMAX
Attention Layer, 5x1
Hidden Layer, 5x2400
Input Layer, 5x3461
Past actions for each customer
Input Layer, (800, 800, 800)
(Weighted Average)
Same per customer, Different per timestep
Hidden Layer, 5x(3x800)
GUMBEL SOFTMAX
Same per customer

Figure 3: Neural Network system (from Nanavati et al. [2020]): (left) Attention Network and Interaction Network to learn shopkeeper-customer interaction logic. (right) Causal Inference Network to learn the causal relation between customer and shopkeeper actions.

For the multi-party interaction case of SPRING, related approaches will be explored, using Deep RL models to train, from human interaction data, an agent to learn the high-level multi-party interaction logic.

4 Design of neural architectures for conversational systems

There are several relevant routes for developing neural components within the overall SPRING conversational system. The initial point to note here is that pure neural architectures for conversational systems, i.e. the recent trend in massive pre-trained Language Models which map input word sequences to output word sequences, in a similar manner to Machine Translation systems, e.g. [Zhang et al., 2020, Radford et al., 2019, Brown et al., 2020], are not in themselves suitable for deployed real-world applications. Firstly, there are many reported examples [Dusek et al., 2020] of such systems, being inconsistent, "hallucinating" information which can be incorrect, making errors of deletion, and substitution, and being repetitive. Their outputs are also generally not conditioned on external representations such as knowledge bases or task representations, as would be required for a deployed HRI system, although there is current research in these areas. Moreover, there are serious ethical issues around the use of neural response generation systems, where biases in training data have been shown to lead to problematic system output including abusive language, hate speech, gender bias, microaggressions, dehumanization, and various socio-political framing biases that are prevalent in language data Bender et al. [2021]. There have been some attempts to remedy this [Xu et al., 2020].

These deficiencies render stand-alone neural conversational / response generation systems inappropriate as solutions to conversational AI for HRI, at least for now. Rather, neural models of various types can be used within larger more semantically controllable conversational AI systems in ways which we describe below.

For example, the current Alana system [Curry et al., 2018, Papaioannou et al., 2017b,c], which the SPRING conversational system extends, already uses neural methods for intent classification and system utterance ranking [Shalyminov et al., 2018]. These components can be extended and further developed to cover the SPRING use-cases and in particular the challenges of multi-user conversation.

There are also several additional directions to explore for the neural conversational architecture in SPRING:

- 1) Deep Reinforcement Learning for action selection in multi-modal multi-user social situations.

- 2) Transformer models to further support unscripted open-domain conversation and chit-chat (e.g. [Zhang et al., 2020, Radford et al., 2019, Brown et al., 2020]),
- 3) Transfer Learning for bootstrapping task-based systems from small data. [Shalyminov et al., 2019]
- 4) Multimodal deep learning for vision and language grounding in HRI [Suglia et al., 2020a].
- 5) Neural methods for NLU, in particular for robust frame-semantic parsing. [Vanzo et al., 2019]

We will briefly elaborate on each of these research directions.

4.1 Deep Reinforcement Learning for multi-modal social situations

As discussed in section 3.2, decision-making for multimodal social situation involves a very high-dimensional state space. In contrast to previous spoken dialogue systems which had task-based state spaces generated by a discrete few slots and values (usually modelled as Markov Decision Processes), or with some continuous dimensions (such as ASR confidence scores in POMDP systems) Rieser and Lemon [2011], conversational systems which have to take into account multimodal signals from multiple users will have much larger state spaces, formed of combinations of continuous and discrete variables. Moreover, where states representing utterance meanings are formed using word-embeddings (as in current neural approaches to dialogue), state spaces again become very large and the decision-making problem becomes intractable without some form of state-space compression [Crook and Lemon, 2011]. Deep RL methods provide a useful approach in such contexts because they can effectively learn a useful compression of the state space while optimizing action selection for a particular reward function (ie. a goal specification).

Deep RL has been used by Li et al. [2016] to optimize an open-domain chat-based dialogue system, where reward was given for ease of answering, Information Flow, and semantic coherence – each of which is a function of utterance embeddings (e.g. Information Flow penalizes semantic similarity between utterances of the agent, so as to avoid repetitive content). Other uses of Deep RL are in utterance selection from an ensemble of bots which produce multiple possible responses in parallel Shalyminov et al. [2018] .

In the multimodal, social, multi-user case of SPRING, Deep RL methods can be used in similar ways. However, the computational resources needed to train such models may be prohibitive, so any use of Deep RL methods will need to be carefully targeted to specific decision problems so that the action space for exploration is reduced.

4.2 Transformer models for open-domain and task-based conversation

Much recent attention has focused on Transformer-based models such as Zhang et al. [2020], Radford et al. [2019], Brown et al. [2020], which are able to produce natural and fluent output based on previous conversational turns. However, as noted above, they encounter problems of semantic consistency, repetition, coherence, and ethical compliance. For these reasons we will explore using such bots as a “fall-back” when other parts of the conversational system have no response to the user’s input. This is likely to be especially useful for social ‘chit-chat’. A related avenue to explore is the semantic parsing of the output of such neural models, so that the overall system maintains its representation of what has been said, for example what topics and entities a neural model has introduced into a conversation. Finally, we will explore to what extent the output of such a model can be conditioned on the conversational context, for example the tasks and objects under discussion [Wen et al., 2015, Peng et al., 2020].

4.3 Transfer Learning for bootstrapping task-based systems

Related to the above discussion, recent work has explored Transfer Learning and approaches which take pre-trained neural Language Models such as [Radford et al., 2019, Brown et al., 2020] and then fine-tune them for specific tasks using only small amounts of data in a target domain [Shalyminov et al., 2019, Peng et al., 2020, Shalyminov et al., 2020]. This is sometimes referred to as ‘bootstrapping’ or ‘few-shot’ learning of a new task-based system, since it creates a new system from a few example conversations in the target task-domain.

For the SPRING project, it will be useful to consider whether new tasks in the hospital reception domain (for example patient check-in) can be developed using such Transfer Learning methods, based on data collected in the project.

4.4 Multimodal deep learning for vision and language grounding

Another deficiency of current neural models is that their representations of word-meaning are not grounded in visual information. This is crucial for HRI in cases where humans wish to talk about or refer to objects in virtue of their visual properties such as color, shape etc. as well as to hold coherent conversations with robots in visual/spatial situations where objects and their relations are critical for task success (e.g. directions to a place or object). In this regard, there has been much recent work on 'visual dialogue' tasks where grounded language understanding and generation in context is paramount Suglia et al. [2020a,b, 2021]. For SPRING, it will be useful to explore to what extent we can use more visually grounded semantic representations of utterances within the multimodal dialogue context developed in the system. Our recent video demonstration at HRI 2021 [Part et al., 2021] [3] is a step in this direction as it uses deep learning for object recognition to create spatial representations which are used to answer questions from the user.

4.5 Neural methods for Natural Language Understanding (NLU)

The need for handling complex and noisy linguistic phenomena which might result in poor recall is particularly evident when dealing with spoken language, which is rich in ungrammatical sentences. To overcome these issues and to handle complex sentences while providing richer semantic representations, frame semantics using deep learning architectures can be employed Vanzo et al. [2019].

HERMIT Vanzo et al. [2019] captures the user intent using semantic parsing Wilks and Fass [1992] where semantic parsing is the process of extracting structured meaning representations from Natural Language inputs. Frame Semantics Fillmore [1985] offers a theory of meaning representation whose basic object is the semantic frame, a conceptual structure representing a situation of the world conveyed in a sentence. Each semantic frame is characterized by a set of frame elements, which are the linguistic arguments enriching the main meaning expressed by the related frame. For example, the sentence "Where can I find a chair?" expresses a LOCATING frame, evoked by the verb *find* which plays the special role of LEXICAL_UNIT. *I* and *a chair* are the PERCEIVER and SOUGHT_ENTITY frame elements, respectively. The HERMIT parsing process is modelled as a cascade of three tasks, each of which is modelled as a Sequence-to-Sequence problem, where a task-specific label is assigned to each token of the sentence according to the IOB2 notation Ramshaw and Marcus [1995]. The network is composed of three blocks of encoding layers, one per task. The first encoding step of each task is realized with a Bi-LSTM Schuster and Paliwal [1997], while an additional Self-Attention layer Li et al. [2018] is added. This latter layer enriches the encoded representation with latent relationships between the words composing a sentence. The final sequence of labels for each task over a sentence is obtained through a Conditional Random Field layer Lafferty et al. [2001] fed with the previous encoding. As well as obtaining state-of-the-art results on the frame-semantic parsing tasks, this system has also been shown to work well on automatically and grammatically incorrect translated sentences (for example translated from Finnish to English) using Machine Translation approaches.

For the SPRING project it will be useful to explore the possibility of using a related approach especially when dealing with translations from French to English.

References

Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL https://www.ros.org.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

R.E. Bellman. *Dynamic programming.* Princeton University Press, 1957.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., 1994. ISBN 0471619779.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge England, 1989.

[3] https://www.youtube.com/watch?v=eY_BNxwrlPg

Christopher John Cornish Hellaby Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

Marc Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2011. ISSN 1935-8253.

Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

Alessandro Lazaric. Transfer in reinforcement learning: A framework and a survey. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 143–173. Springer, 2012.

Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*, 2016.

Yuan Gao, Fangkai Yang, Martin Frisk, Daniel Hemandez, Christopher Peters, and Ginevra Castellano. Learning socially appropriate robot approaching behavior toward groups using deep reinforcement learning. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8. IEEE, 2019.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, and Jiakai Zhang. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Eduardo F. Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

Adam Kendon. Spacing and orientation in co-present interaction. In *Development of multimodal interfaces: Active listening and synchrony*, pages 1–15. Springer, 2010.

Xuan-Tung Truong and Trung-Dung Ngo. “To approach humans?”: A unified framework for approaching pose prediction and socially aware robot navigation. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):557–572, 2017. Publisher: IEEE.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Diogo Rato, Samuel Mascarenhas, and Rui Prada. Towards social identity in socio-cognitive agents. *ArXiv*, 2020.

Daniel Hernández García, Yanchao Yu, Weronika Sieinska, Jose L. Part, Nancie Gunson, Oliver Lemon, and Christian Dondrup. Explainable representations of the social state: A model for social human-robot interactions. *CoRR*, abs/2010.04570, 2020. URL https://arxiv.org/abs/2010.04570.

Dan Bohus and Eric Horvitz. *Situated Interaction*, page 105–143. Association for Computing Machinery and Morgan & Claypool, 2019. ISBN 9781970001754. URL https://doi.org/10.1145/3233795.3233800.

Dan Bohus and Eric Horvitz. Models for multiparty engagement in open-world dialog. In *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, SIGDIAL ’09, page 225–234, USA, 2009. Association for Computational Linguistics. ISBN 9781932432640.

Marta Romeo, Daniel Hernandez Garcia, Ray Jones, and Angelo Cangelosi. Deploying a deep learning agent for hri with potential “end-users” at multiple sheltered housing sites. In *7th International Conference on Human-Agent Interaction*, July 2019. 7th International Conference on Human-Agent Interaction, HAI 2019 ; Conference date: 06-10-2019 Through 10-10-2019.

Jose Part, Daniel Hernandez-Garcia, Yanchao Yu, Nancie Gunson, Christian Dondrup, and Oliver Lemon. Towards visual dialogue for human-robot interaction. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, 2021.

Ioannis Papaioannou, Christian Dondrup, Jekaterina Novikova, and Oliver Lemon. Hybrid chat and task dialogue for more engaging HRI using reinforcement learning. In *26th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2017, Lisbon, Portugal, August 28 - Sept. 1, 2017*, pages 593–598. IEEE, 2017a. doi: 10.1109/ROMAN.2017.8172363. URL https://doi.org/10.1109/ROMAN.2017.8172363.

Klaus Weber, Hannes Ritschel, Ilhan Aslan, Florian Lingenfelser, and Elisabeth André. How to shape the humor of a robot - social behavior adaptation based on reinforcement learning. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction*, ICMI ’18, page 154–162, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356923. doi: 10.1145/3242969.3242976. URL https://doi.org/10.1145/3242969.3242976.

A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro. Robot gains social intelligence through multimodal deep reinforcement learning. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 745–751, 2016. doi: 10.1109/HUMANOIDS.2016.7803357.

A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro. Show, attend and interact: Perceivable human-robot social interaction through neural attention q-network. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1639–1645, 2017. doi: 10.1109/ICRA.2017.7989193.

Heriberto Cuayáhuitl. A data-efficient deep learning approach for deployable multimodal social robots. *Neurocomputing*, 396:587–598, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.09.104. URL https://www.sciencedirect.com/science/article/pii/S092523121930493X.

Simon Keizer, Mary Ellen Foster, Oliver Lemon, Andre Gaschler, and Manuel Giuliani. Training and evaluation of an mdp model for social multi-user human-robot interaction. In *Proceedings of the 14th Annual SIGdial Meeting on Discourse and Dialogue*, August 2013.

Amal Nanavati, Malcolm Doering, Dražen Brščić, and Takayuki Kanda. Autonomously learning one-to-many social interaction logic from human-human interaction data. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, HRI ’20, page 419–427, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367462. doi: 10.1145/3319502.3374798. URL https://doi.org/10.1145/3319502.3374798.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. Dialogpt: Large-scale generative pre-training for conversational response generation, 2020.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Ondrej Dusek, Jekaterina Novikova, and Verena Rieser. Evaluating the state-of-the-art of end-to-end natural language generation: The e2e nlg challenge. *Computer Speech & Language*, 59:123–156, 2020. ISSN 0885-2308. doi: https://doi.org/10.1016/j.csl.2019.06.009. URL https://www.sciencedirect.com/science/article/pii/S0885230819300919.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Margaret Mitchell. On the dangers of stochastic parrots: Can language models be too big. 2021.

Jing Xu, Da Ju, Margaret Li, Y-Lan Boureau, Jason Weston, and Emily Dinan. Recipes for safety in open-domain chatbots. *arXiv preprint arXiv:2010.07079*, 2020.

Amanda Cercas Curry, Ioannis Papaioannou, Alessandro Suglia, Shubham Agarwal, Igor Shalyminov, Xinnuo Xu, Ondřej Dušek, Arash Eshghi, Ioannis Konstas, Verena Rieser, et al. Alana v2: Entertaining and informative open-domain social dialogue using ontologies and entity linking. *Alexa Prize Proceedings*, 2018.

Ioannis Papaioannou, Amanda Cercas Curry, Jose Part, Igor Shalyminov, Xu Xinnuo, Yanchao Yu, Ondřej Dušek, Verena Rieser, and Oliver Lemon. An ensemble model with ranking for social dialogue. 2017b. NIPS 2017 Conversational AI Workshop; Conference date: 08-12-2017 Through 08-12-2017.

Ioannis Papaioannou, Amanda Cercas Curry, Jose L. Part, Igor Shalyminov, Xinnuo Xu, Yanchao Yu, Ondřej Dušek, Verena Rieser, et al. Alana: Social dialogue using an ensemble model and a ranker trained on user feedback. *Alexa Prize Proceedings*, 2017c.

Igor Shalyminov, Ondřej Dušek, and Oliver Lemon. Neural response ranking for social conversation: A data-efficient approach. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*, pages 1–8, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5701.

Igor Shalyminov, Sungjin Lee, Arash Eshghi, and Oliver Lemon. Few-shot dialogue generation without annotated data: A transfer learning approach, 2019.

Alessandro Suglia, Ioannis Konstas, Andrea Vanzo, Emanuele Bastianelli, Desmond Elliott, Stella Frank, and Oliver Lemon. Compguesswhat?!: A multi-task evaluation framework for grounded language learning, 2020a.

Andrea Vanzo, Emanuele Bastianelli, and Oliver Lemon. Hierarchical multi-task natural language understanding for cross-domain conversational ai: Hermit nlu. *SIGDIAL*, 2019.

Verena Rieser and Oliver Lemon. Learning and evaluation of dialogue strategies for new applications: Empirical methods for optimization from small data sets. *Computational Linguistics*, 37(1):153–196, 2011.

Paul A. Crook and Oliver Lemon. Lossless Value Directed Compression of Complex User Goal States for Statistical Spoken Dialogue Systems. In *Proceedings of Interspeech*, 2011.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation, 2016.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems, 2015.

Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. Few-shot natural language generation for task-oriented dialog, 2020.

Igor Shalyminov, Alessandro Sordoni, Adam Atkinson, and Hannes Schulz. Hybrid generative-retrieval transformers for dialogue domain adaptation, 2020.

Alessandro Suglia, Antonio Vergari, Ioannis Konstas, Yonatan Bisk, Emanuele Bastianelli, Andrea Vanzo, and Oliver Lemon. Imagining grounded conceptual representations from perceptual information in situated guessing games, 2020b.

Alessandro Suglia, Yonatan Bisk, Ioannis Konstas, Antonio Vergari, Emanuele Bastianelli, Andrea Vanzo, and Oliver Lemon. An empirical study on the generalization power of neural representations learned via visual guessing games, 2021.

Yorick Wilks and Dann Fass. The preference semantics family. *Computers & Mathematics with Applications*, 23(2):205 – 221, 1992. ISSN 0898-1221. doi: https://doi.org/10.1016/0898-1221(92)90141-4. URL http://www.sciencedirect.com/science/article/pii/0898122192901414.

Charles J. Fillmore. Frames and the semantics of understanding. *Quaderni di Semantica*, 6(2):222–254, 1985.

Lance Ramshaw and Mitch Marcus. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*, 1995. URL https://www.aclweb.org/anthology/W95-0107.

M. Schuster and K.K. Paliwal. Bidirectional Recurrent Neural Networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997. ISSN 1053-587X. doi: 10.1109/78.650093. URL http://dx.doi.org/10.1109/78.650093.

Changliang Li, Liang Li, and Ji Qi. A self-attentive model with gate mechanism for spoken language understanding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3824–3833. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/D18-1417.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.655813.