Deliverable D6.5: Final Neural Network Architectures

Due Date: 08/11/2022

Main Author: INRIA

Contributors: BIU, HWU, UNITN

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	D6.5: Final Neural Network Architectures
Responsible Partner	INRIA
Work Package	WP6: Learning Robot Behaviour
Task	T6.1: Deep Architectures for Conversational System and Non-verbal Behaviour
Version & Date	08/11/2022
Dissemination	Public Deliverable

CONTRIBUTORS AND HISTORY

Version	Editor	Date	Change Log
1.0	BIU, HWU, INRIA, UNITN	18/10/2022	
1.1	BIU, HWU, INRIA	07/11/2022	minor changes based on internal review
1.2	HWU	08/11/2022	minor changes based on 2. internal review

APPROVALS

Authors/editors	BIU: Sharon Gannot; HWU: Angus Addlesee, Daniel Hernandez Garcia, Oliver Lemon; INRIA: Xavier Alameda-Pineda, Anand Ballou, Chris Reinke; UNITN: Bin Ren, Elisa Ricci
Task Leader	INRIA
WP Leader	INRIA

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

Contents

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

Abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
ARI	Social assistive robot used by the SPRING project
API	Application Programming Interface
BIU	Bar-Ilan University (SPRING Partner)
CNN	Convolutional Neural Network
CVUT	Czech technical university in Prague (SPRING Partner)
DRL	Deep Reinforcement Learning
DNN	Deep Neural Network
GAN	Generative Adversarial Network
GPI	General Policy Iteration
GRU	Gated Recurrent Units
FGD	Fréchet Gesture Distance
HRI	Human Robot Interaction
HWU	Heriot-Watt University (SPRING Partner)
INRIA	Institut National de Recherche en sciences et technologies du numérique (SPRING Partner)
MAE	Mean Average Error
MAEJ	Mean Average Error of Joint coordinates
MDP	Markov Decision Process
MPC	Multi Party Conversation
PAL	PAL Robotics (SPRING Partner)
RelTR	Relation Transformer
RL	Reinforcement Learning
ROS	Robot Operating System
RBF	Radial Basis Function
SPRING	Socially Pertinent Robots in Gerontological Healthcare
SF	Successor Feature
SFR	Successor Feature Representation
SR	Successor Representation
UNITN	University of Trento (SPRING Partner)
WP	Work Package (of the SPRING project)

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

Executive Summary

This deliverable reports the results of SPRING task T6.1: "Deep Architectures for Conversational System and Non-verbal Behaviour". It focuses on the neural architectures and related methods that are developed for the non-verbal control of the ARI robot as part of WP6: "Robot Learning Behavior". It furthermore summarizes the neural architectures that were developed for WP2: "Environment Mapping, Self-localisation and Simulation", WP3: "Robust Audio-visual Perception of Humans", WP4: "Multi-Modal Human Behaviour Understanding", and WP5: "Multi-User Spoken Conversations with Robots".

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

1 Introduction

The design of neural architectures is the goal of SPRING task T6.1: "Deep Architectures for Conversational System and Non-verbal Behaviour" and part of WP 6: "Robot Learning Behavior". The goal of task T6.1 is to design neural architectures for deep reinforcement learning and associated methods, that should serve as the methodological basis for all modules learning and using optimal decision policies (T5.2, T5.3, T6.3). INRIA and HWU investigated network designs for the low-level behaviour system, the task planner, natural language understanding and generation, and the dialogue and non-verbal behaviour managers. Moreover, CVUT, BIU and UNITN designed network streams responsible for exploiting the sensory data issued from WP2, WP3, and WP4, respectively. Please note, as deep neural architecture play a vital role for many components of the SPRING project, they will be further refined and developed. This document represents their current developmental state. All developed software that is part of Task T6.1 is open source, and made available open access to the public on the project repository until at least 4 years after the project ends.

This deliverable summarizes the developed neural architectures and related methods. It is structured by WP for which the neural architectures were developed. Many architectures have been already described in existing SPRING deliverables. For these cases, the architectures are only shortly summarized with a reference to the deliverable that gives a more in-depth description. The neural architectures investigated and developed by CVUT for WP2 ("Environment Mapping, Self-localisation and Simulation") are described in detail in SPRING Deliverable 2.1 [43] and 2.2 [44]. The main focus of this deliverable is on the work in WP6 which is described in greater detail in Chapter 5.

2 WP3 - Robust Audio-Visual Perception of Humans

Objectives in WP3 are responsible for the robust extraction of users' low-level characteristics from the raw auditory and visual data. Characteristics include positions, speaking status and speech signals. The following two sections describe first deep neural networks developed for the visual localization and tracking of humans, followed by an overview over the networks used for the audio processing pipeline.

2.1 Visual Localization and Tracking of Humans

(Contributor: INRIA)

The goal of this neural architecture is to detect, identify, and track people using visual data from the front fish eye camera. The proposed multi-person localisation and tracking module (Fig. 2.1) is based on FairMOT [67] and is described in more detail in SPRING Deliverable 3.1 [45] and 3.2 [46]. It combines the CenterNet object detection architecture [70] with Deep Layer Aggregation [63]. The latter is a general framework that can be used to augment a backbone architecture of choice to better fuse information among layers. The network uses as a backbone the ResNet34 [20] network. In difference to the original FairMOT architecture, we used a smaller resolution of 800x448 for the input images to improve computational performance. Another modification is the increase in the dimension of the Re-Identification branch from 128 as in the original FairMOT to 512. This improved the robustness of the tracker, especially to identity switches and in situations where people were leaving the scene and re-entering a while after.

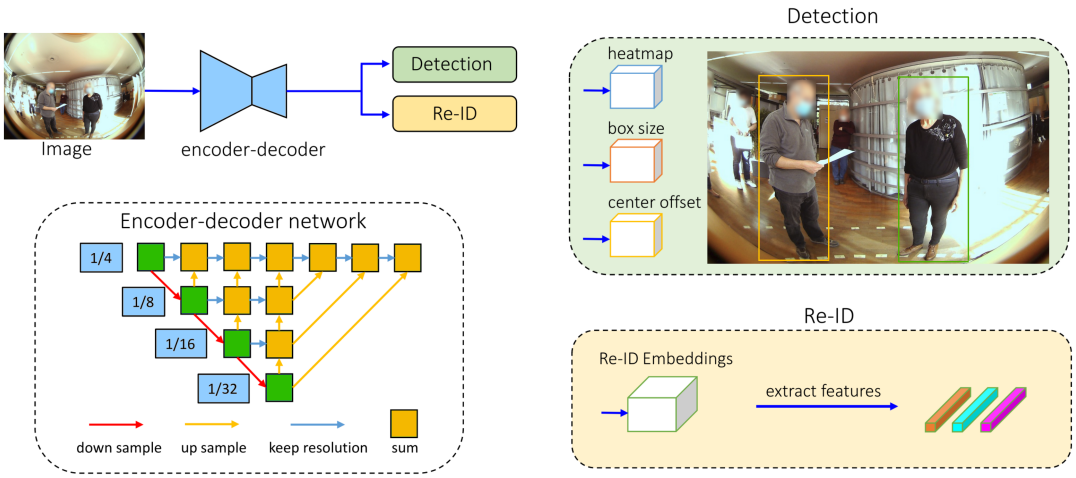


Figure 2.1: Architecture of FairMOT used for the localization and tracking of people. The input image is processed by a ResNet encoder-decoder network. Its output is used for the detection of people, i.e. the bounding boxes around them, and the Re-Identification which gives them a consistent ID over several time frames even if occlusions occur. (Figure adapted from [67].)

2.2 Audio Processing Pipeline

(Contributor: BIU)

The goal of the audio processing pipeline is to acquire the sound signal from the ReSpeaker audio device, enhance it, and then apply an automatic speech recognition algorithm to transcribe the audio utterances for further processing by the dialogue system. The audio pipeline is depicted in Fig. 2.2.

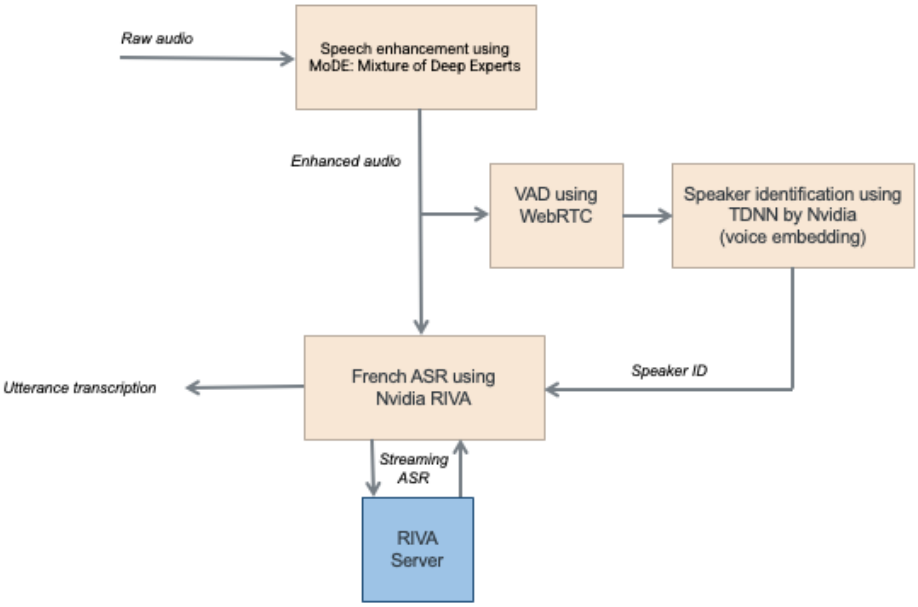


Figure 2.2: Audio pipeline.

The audio signal is acquired by the ReSpeaker 4-channel microphone array on ARI. The first step of the audio processing pipeline is the application of a speech enhancement algorithm. We are applying a Mixture of Deep Experts (MoDE) architecture (developed by BIU) that uses a single microphone. The algorithm is capable of significantly suppressing background noise, e.g., babble noise [8]. At the later stage of the project we will also incorporate speaker separation (either multi-microphone or single-microphone), and a narrow-band noise reduction module. If required, we will also implement a dereverberation algorithm. Echo cancellation (to supress ARI's self-voice) will be incorporated, if a full-duplex dialogue management system will be deployed.

The enhanced speech is fed into automatic speech recognition (ASR) engine in the French language. We use a pre-trained RIVA engine, which is a conformer-based, on-prem streaming solution by Nvidia.[1] The transcribed utterance is fed to the dialogue system.

In parallel, we apply the speaker identification step, comprising two stages. In the first stage, we apply a voice activity detector (VAD) using the WebRTC package.[2] In the second stage, voice representation is extracted using a pre-trained TitaNet model by Nvidia [23]. In the last step of the pipeline, the speaker ID is merged with the corresponding transcribed utterance. In the future, the location of the speaker, as extracted from the audio-visual data, will be associated as well.

[1] https://developer.nvidia.com/riva
[2] https://webrtc.org/

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

3 WP4 - Multi-Modal Human Behaviour Understanding

We developed 5 modules as part of WP4 ("Multi-Modal Human Behaviour Understanding"):

- Biometric recognition
- Monocular depth estimation
- Multi-target body pose estimation
- Gaze target recognition
- Face mask detection

Since we already provided the detailed description of the neural architectures in terms of all the modules mentioned above in D4.1 [47], in this deliverable, we briefly list out the corresponding parts in the following sections.

3.1 Biometric Recognition

(Contributor: UNITN)

The task of biometric recognition is to detect faces, identify if they have been already seen, and the classification of age and gender. We developed a transfer method to utilize in a more efficient manner FaceNet [42], a state-of-the-art deep convolutional network that learns latent embeddings for faces. These embeddings are then used as input for networks of the downstream tasks (identification, classification of age and gender). The problem with FaceNet is that it is very large, having 27.9 million parameters, so that it is very time consuming to be executed. We implemented a transfer method to train a smaller network, having only 6.7 million parameters, but that reaches the same performance as FaceNet. Please refer to D4.1 [47] (Section 2.4) for more details.

3.2 Monocular Depth Estimation

(Contributor: UNITN)

The monocular depth estimation is based on a state-of-the-art network designed for embedded systems: FastDepth [60]. Such network has been improved by replacing the MobileNet backbone with the latest version of MobileNetV3 Large [21]. Our implementation is capable of dense depth predictions while running on low-powered CPUs in real-time. Some qualitative results of our monocular depth estimation network is available on D4.2 [48] (Section 3).

3.3 Multi-Target Body Pose Estimation

(Contributor: UNITN)

The pose estimation detects the joints and limbs of people in the fish eye camera view of ARI. For this purpose we integrated OpenPose[1] which is based on a two-branch multi-stage CNN [7]. Please refer to D4.1 [47] (Section 2.5) for more details.

[1] https://github.com/CMU-Perceptual-Computing-Lab/openpose

3.4 Gaze Target Recognition

(Contributor: UNITN)

Our multimodal pipeline for gaze target detection has three-pathways to process shown in Fig 3.1: the head image, the scene image, and the depth map. We implicitly learn the orientation features using the head attention network. The RGB and depth features obtained by the scene and depth networks are multiplied by the head attention map, which allows the fusion and prediction module to focus on objects in the field-of-view of the person of interest. The multimodal joint learning and gaze prediction are performed by the fusion and prediction component via late-fusion of the RGB and depth embeddings. The outputs of our architecture are: 2D heatmap, where higher values are closer to the gaze target, and the probability of the gaze being inside or outside the scene. A set of qualitative results of our gaze target estimation network is available on D4.2 [48] (Section 2.3).

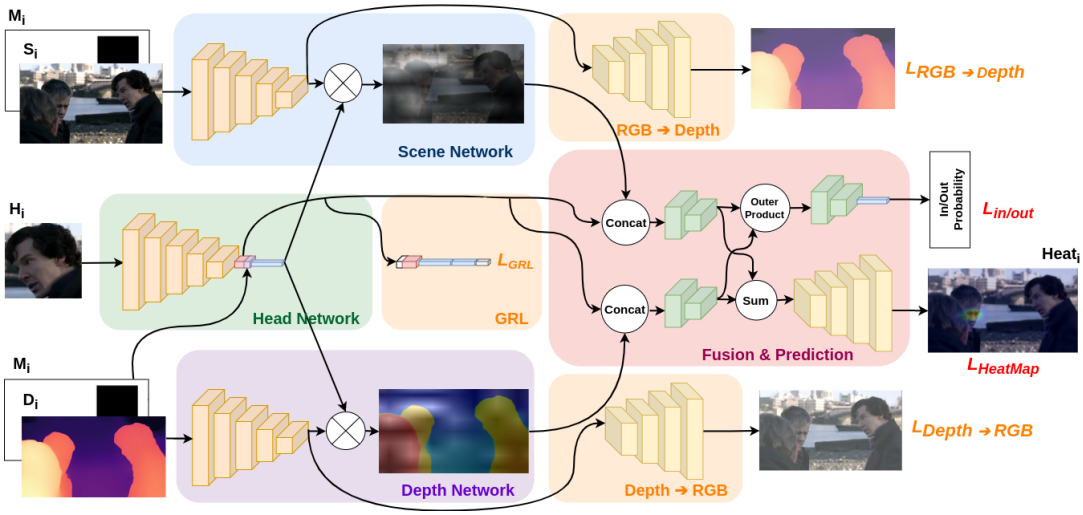


Figure 3.1: Architecture of the gaze target detection.

3.5 Face Mask Detection

(Contributor: UNITN)

We implemented a deep neural network face detection mechanism. We started from the open source API provided by AIZOOTech[2]. For the ROS integration, a ROS wrapper was developed by us for integration. The prototype of the mask detection model is a Single Shot MultiBox Detector (SSD) [31]. The model architecture is shown in Fig. 3.2. Please refer to D4.1 [47] (Section 2.3) for more information.

[2] https://github.com/AIZOOTech/FaceMaskDetection

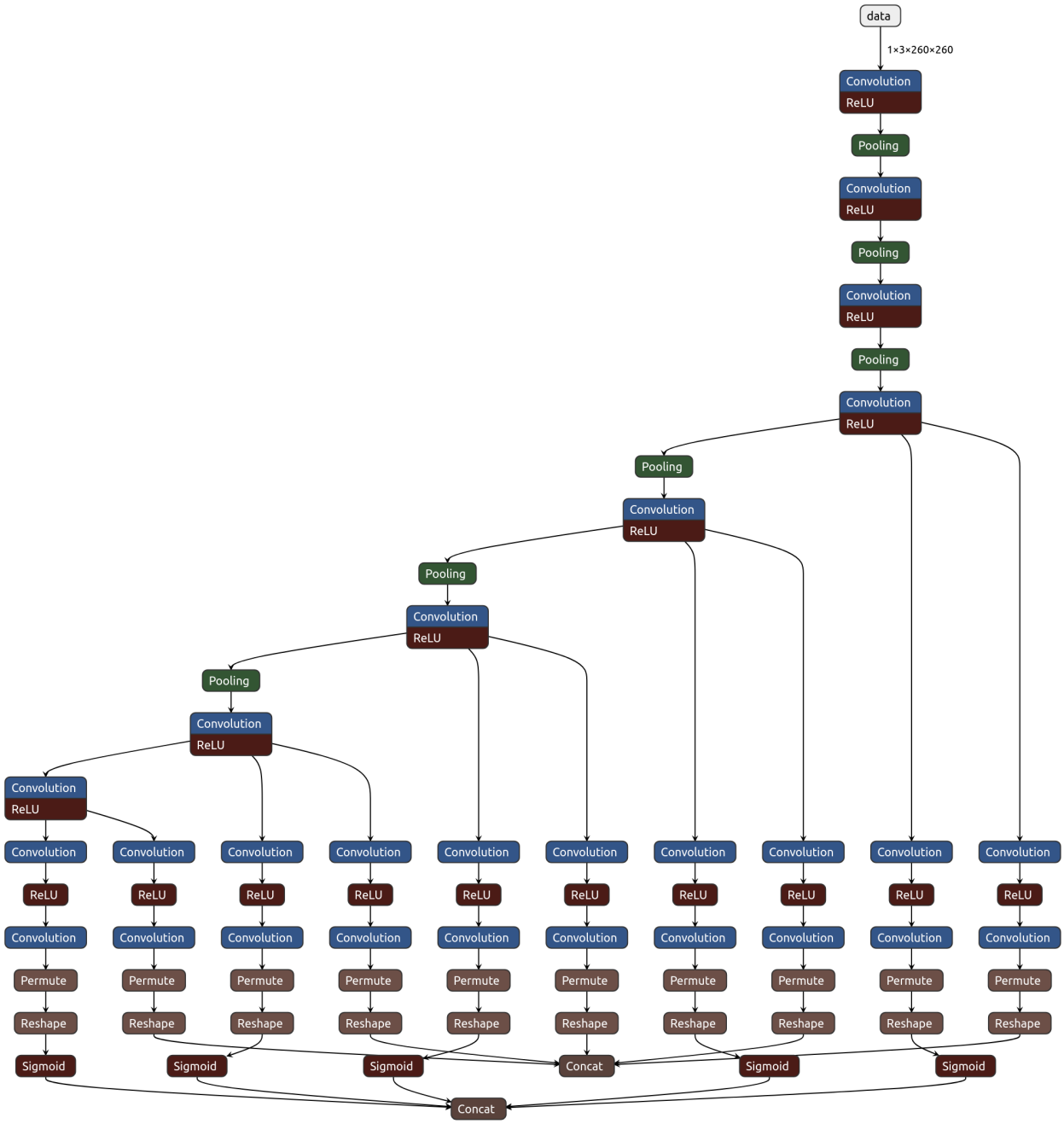


Figure 3.2: Mask detection model which is based on the Single Shot MultiBox Detector [31]. The image is from `https://github.com/AIZOOTech/FaceMaskDetection`.

4 WP5 - Multi-User Spoken Conversations with Robots

Several neural architectures and techniques for multi-user conversation involving a robot and multiple humans are developed to train the conversational system, including natural language understanding, dialogue management, and natural language generation modules. To deal with the challenges of a multi-user conversation AI system for Human-Robot Interaction several neural conversational architectures were developed for SPRING:

- Neural Networks for Multi-Party Conversation Understanding
- Multi-modal Deep Learning for Vision and Language Grounding

The current 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 [6, 37, 68], are not in themselves suitable for deploying real-world applications, especially in a social HRI setting as SPRING, as there are many reported examples [14] of such systems, being inconsistent, 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.

These deficiencies render stand-alone neural conversational / response generation systems inappropriate as solutions to conversational AI for HRI. However, neural models of various types can be used within a larger more semantically controllable conversational AI system as we are developing in the multi-party conversational system of SPRING.

4.1 Language Models for Multi-Party Conversations

(Contributor: HWU)

Multi-Party Conversations (MPC), where a computational agent needs to interact with two or more humans, cannot simply be modelled as several bi-lateral systems running in parallel, since there are important interactions between the humans which the system will need to track accurately.

Most of the existing methods focus on building dialogue systems that assume utterances are sequential but this is not the case in multi-party conversations. Utterances in a two-party conversation alternate between interlocutors in a sequential information flow. However, in a MPC each utterance can be spoken by anyone and address anyone else in the conversation, which constitutes a graphical information flow [17], see Figure 4.1.

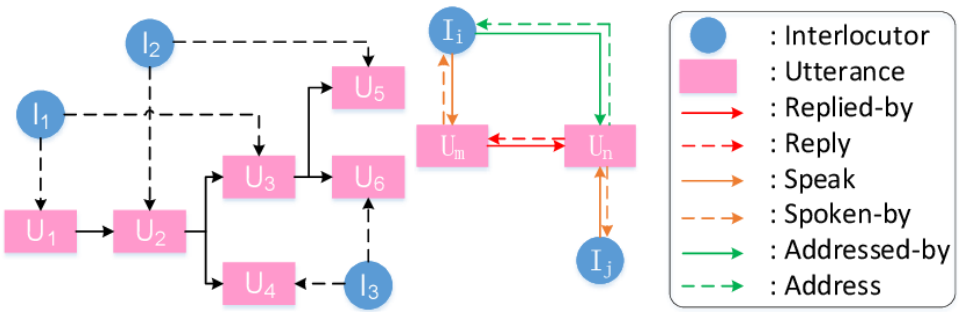


Figure 4.1: Illustration of a graphical information flow in an MPC. Pink rectangles denote utterances and blue circles denote interlocutors. [16, 17].

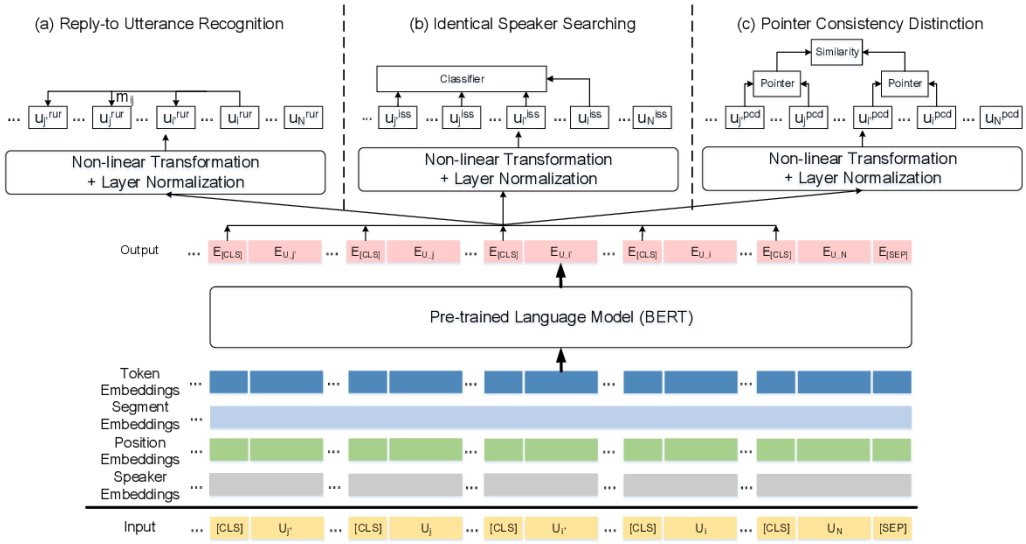


Figure 4.2: Input representations and model architectures of the three self-supervised tasks for interlocutor structure modeling, including (a) reply-to utterance recognition, (b) identical speaker searching and (c) pointer consistency distinction in MPC-BERT [18].

Models for Multi-Party Conversation Understanding

There are several important research problems to be addressed in MPC understanding: addressee identification; incrementality; multi-party state tracking; turn-taking; and response generation. Multi-party state tracking needs to maintain an accurate representation of each individual agent's goals, plans, and information. While intuitively, the complicated interactions between interlocutors, between utterances, and between interlocutors and utterances might make these tasks complementary among each other, most of existing studies design models for each individual task in MPCs separately [17].

The work of Gu et al. [18], proposed first a unified pre-trained model. The model is unified as it represents both utterances and interlocutors together, with several supervised tasks for pre-training a language model focusing on modeling only interlocutor structures and utterance semantics. It jointly learns "who says what to whom", Addressee recognition, speaker identification, and response selection by considering addressee and speaker related tasks as complementary signals for response selection during the pre-training stage. Their model, named MPC-BERT, choose BERT [13] as the backbone of their pre-trained language model for MPC. A task-dependent non-linear transformation layer is placed on top of BERT in order to adapt the output of BERT to different tasks, the architecture can be seen on Figure 4.2.

[69] proposed DialogLM, a pre-training framework for long dialogue understanding and summarization. Considering the nature of MPCs, they propose dialogue window denoising approach for generatively pre-training DialogLM. Through window-based denoising, the model can learn the format and characteristics of dialogues in a general way, thereby performing better in various dialogue-oriented tasks. Furthermore, to process longer input, they augment the model with sparse attention which is combined with conventional attention in a hybrid manner. DialogLM, [69],

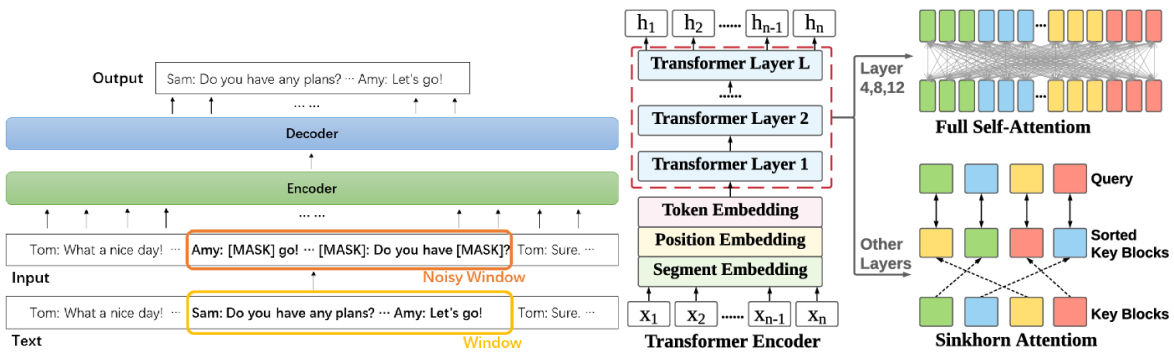


Figure 4.3: Pre-train task for DialogLM: window-based denoising. And model architecture for DialogLM: introducing a hybrid attention approach in the Transformer architecture [69].

chooses a Transformer as their backbone neural architecture, because it shows promising performance and flexibility on various NLP tasks, but leverages hybrid attention method in the Transformer architecture with the recently proposed sparse Sinkhorn attention [56], as shown in Figure 4.3.

We will use our recently designed multi-party wizard-of-oz data collection with 2 humans and an ARI robot, [50], for pre-training and fine-tuning a T5 model [38] on DialogLM [69]. The goal of WP5 and Task 5.3 is to use the collected data to develop a system that is able to track multiple people in conversation: what each speaker is saying, what their individual goals/questions are, etc. As well as correctly tracking the goals of each user it should be able to determine which goals have been met and which still need to be addressed, and determine points in the conversation at which it is possible or desirable to "barge-in" with information meeting a user's goals.

4.2 Semantic Scene Graphs for Visual Grounded Dialogue

(Contributor: HWU)

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, 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 [51, 52, 53]. Visual language grounding refers to the connection between words (symbols) and their referents in the visual space, e.g., objects, their relationships, and their attributes. For SPRING we explore to what extent we can use more visually grounded semantic representations of utterances within the multimodal dialogue context developed in the system, our video demonstration at HRI 2021[1] [34] was a first 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.

Semantic scene graphs are data structures that encode the semantic information of a visual scene into a graph, where nodes represent instances of some sort, and edges represent their relationships. This allows for more compact and interpretable representations that can be leveraged for inference about the scene. Scene graphs give the opportunity to organise the knowledge about a scene into a queryable structure. Helping to build systems that can answer certain questions about the environments where they operate.

Scene Graph Generation

The scene graph generation module is implemented as a ROS action server. Given an empty goal, the server processes the current image from the robot's camera feed and returns the scene graph information by means of a custom message that contains fields for the object bounding boxes (*bbox*), labels (*bbox_labels*) and scores (*bbox_scores*), and for the relationship pairs (*rel_pairs*), their labels (*rel_labels*) and scores (*rel_scores*). By interpreting this information, an action client can produce dialogue responses that leverage the semantic information available in the scene graph.

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

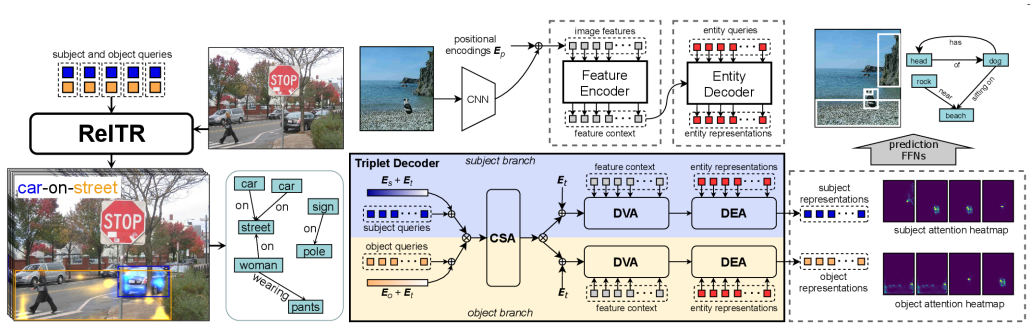


Figure 4.4: RelTR encoder-decoder architecture [11]. A pair of subject and object representations with attention heat maps is decoded into a triplet <subject-predicate-object> by feed forward networks (FFNs). CSA, DVA and DEA stand for Coupled Self-Attention, Decoupled Visual Attention and Decoupled Entity Attention. Ep, Et, Es and Eo are the positional, triplet, subject and object encodings respectively.

The proposed approach generates scene graphs by an end-to-end scene graph generation model Relation Transformer (RelTR) [11]. RelTR (Fig. 4.4) is one-stage end-to-end framework for scene graph generation based on Transformer's encoder-decoder architecture. The encoder reasons about the visual feature context while the decoder infers a fixed-size set of triplets subject-predicate-object using different types of attention mechanisms with coupled subject and object queries. RelTR can predict the subjects (blue), objects (orange) and their predicates simultaneously with learned subject and object queries (Fig. 4.5).

In contrast to most existing scene graph generation methods, [29, 30, 54, 65, 66], RelTR is a one-stage method that predicts sparse scene graphs directly only using visual appearance without combining entities and labeling all possible predicates. The model was trained on Visual Genome [24], a large dataset of images annotated with objects, their attributes and relationships to other objects, natural language descriptions, and question/answer pairs. Compared with other state-of-the-art methods, RelTR achieves state-of-the-art performance using only visual appearance, with very few model parameters and fast inference.

The model for Vision and Language Grounding leverage semantic scene graphs to guide visually grounded conversations (the system architecture is illustrated in Fig. 4.5). Concretely, by having access to a graph representation of the visual scene, the system is able to reason about the location of objects in order to answer user queries as well as to produce meaningful descriptions. Moreover, by combining such ability with task-based and open dialogue conversational system developed in WP5 [49], the system is able to engage in richer conversations with its users.

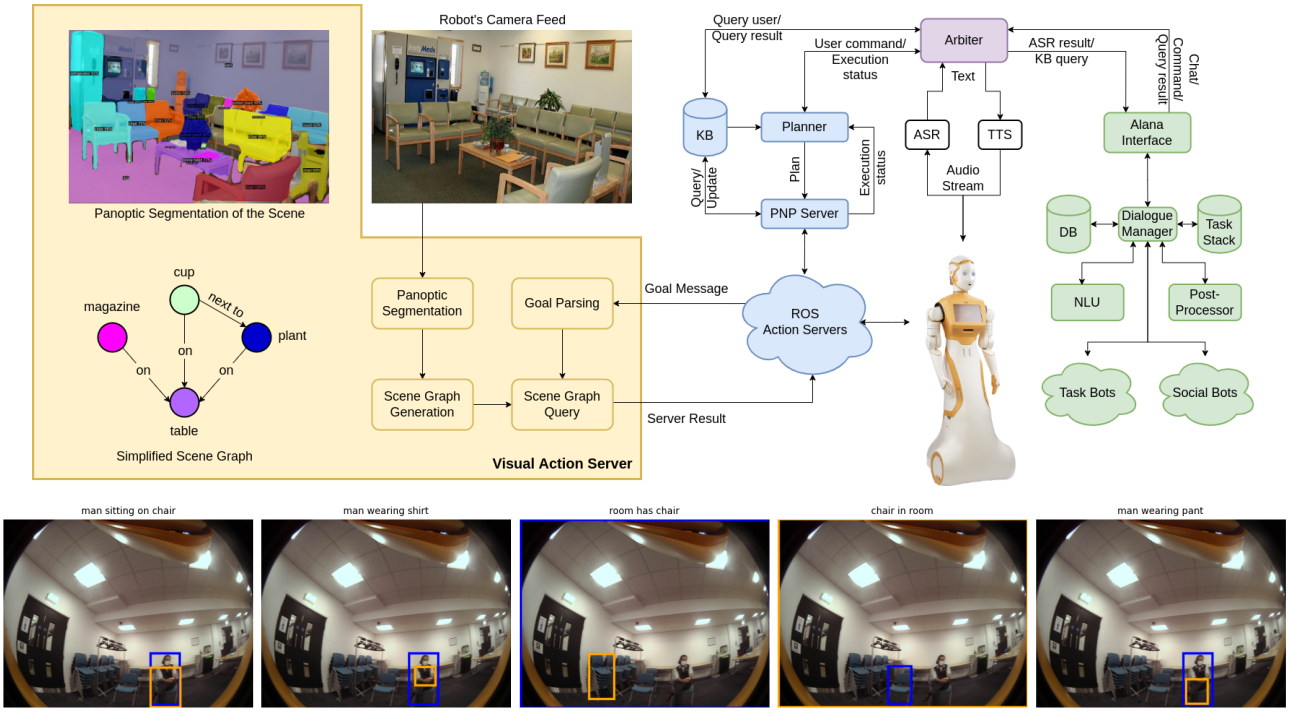


Figure 4.5: Top: Visual grounded situated dialogue combine with the conversation system architecture [49]. Bottom: Visual Dialogue examples with RelTR and ARI robot cameras in a mock-up hospital reception waiting room.

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

5 WP6 - Learning Robot Behaviour

Several neural architectures and methods for an efficient training of neural architectures were developed to learn and control non-verbal robot behaviors. Such behaviors include:

- Human-aware navigation,
- Navigation with the purpose to optimize certain sensor measurements such as to perceive people and listen to their utterances
- Co-speech gesture generation

The next section introduces Transfer and Meta Learning methods to train Deep Reinforcement Learning (DRL) neural architectures. The second section introduces a network to generate gestures for ARI during conversations. The focus of this deliverable is on WP6 because its final results have not yet been discussed in previous deliverables. Therefore, this chapter introduces the algorithms and architectures in more detail compared to previous chapters.

5.1 Transfer & Meta Deep Reinforcement Learning

We investigated two procedures to aid the training of deep neural architectures in the context of Reinforcement Learning (RL). RL provides methods to learn sequential decision making tasks which comprise many of our target behaviors such as: human-aware navigation; joining, keeping and exiting a conversation with a group of people; or the optimal collection of sensory information (visual and auditory). To handle such complex tasks deep neural networks are utilized as function approximators for the central RL components such as value functions or policies. This Deep RL (DRL) approach has the problem of requiring large amounts of data that need to be collected in simulation or on the robot which is time intensive. A second problem, for most of our targeted social robotics tasks, is that the definition of the reward function which is optimized by DRL is often not evident. It needs to be iteratively searched by RL experiments which further increases the need for data. As a solution, we explored Transfer and Meta Learning methods in the context of DRL with the goal to reduce the amount of needed data. The next section introduces the background in DRL and Transfer & Meta Learning, followed by descriptions of the developed methods and their results.

5.1.1 Background: Deep Reinforcement Learning and Transfer & Meta Learning

RL investigates algorithms to solve multi-step decision problems, aiming to maximize the sum over future rewards [55]. RL problems are modeled as *Markov Decision Processes* (MDPs) which are defined as a tuple $M \equiv (\mathcal{S}, \mathcal{A}, p, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action set. An agent transitions from a state s_t to another state s_{t+1} using action a_t at time point t collecting a reward r_t: $s_t \xrightarrow{a_t, r_t} s_{t+1}$. This process is stochastic and the transition probability $p(s_{t+1}|s_t, a_t)$ describes which state s_{t+1} is reached. The reward function R defines the scalar reward $r_t = R(s_{t+1}) \in \mathbb{R}$ for the transition. The goal in an MDP is to maximize the expected return $G_t = \mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k}\right]$, where $R_t = R(S_{t+1})$. The discount factor $\gamma \in [0, 1)$ weights collected rewards by discounting future rewards stronger. RL provides algorithms to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ defining which action to take in which state to maximise G_t.

Value-based RL methods use the concept of value functions to learn the optimal policy. The state-action value function, called Q-function, is defined as the expected future return taking action a_t in s_t and then following policy π:

$$Q^{\pi}(s_t, a_t) = \mathrm{E}_{\pi}\left\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right\} = \mathrm{E}_{\pi}\left\{r_t + \gamma Q^{\pi}(S_{t+1}, A_{t+1})\right\} \ . \tag{5.1}$$

The Q-function can be recursively defined following the Bellman equation such that the current Q-value $Q^{\pi}(s_t, a_t)$ depends on the Q-value of the next state $Q^{\pi}(s_{t+1}, a_{t+1})$. The optimal policy for an MDP can then be expressed based on the Q-function, by taking at every step the maximum action: $\pi^*(s) \in \mathrm{argmax}_a\, Q^*(s, a)$.

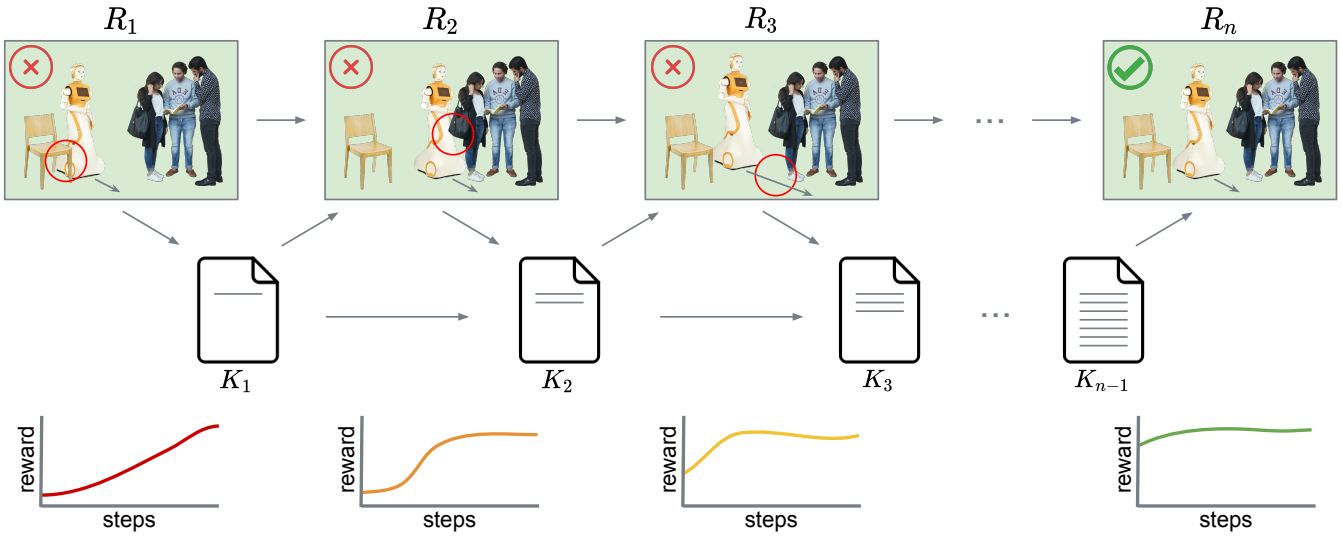


Figure 5.1: Iterative Transfer RL procedure for social robotics. How to define the reward function $R(s)$ in social robotics is often not obvious. For example, the learned behavior for reward function R_1 collides with objects, whereas for R_2 it collides with humans. Several functions $(R_1, \ldots, R_n)$ need to be evaluated before an appropriate function is found that results in the wanted behavior. Evaluating each reward function costs time. Transfer Learning is used to extract and use knowledge K from the training of behaviors for previous reward functions to new reward functions with the goal to improve the learning speed, i.e. to reach a high reward with fewer learning steps.

The optimal Q-function can be learned using a temporal difference method such as Q-learning [59]. Given a transition (s_t, a_t, r_t, s_{t+1}), the Q-value is updated according to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) , \tag{5.2}$$

where $\alpha \in (0, 1]$ is the learning rate.

Deep Reinforcement Learning (DRL) uses deep neural architectures to approximate the central RL components, such as the Q-function (5.1). See [28] for an in-depth introduction to DRL. DRL has the problem that it often needs a lot of training examples, i.e. observations of transitions in the environment, to learn a behavior [25]. This problem is especially aggravated for social robotics, where interactions are costly in terms of time.

Moreover, a further problem in social robotics is the question of how to define the reward function for a targeted behavior [1]. It is often not foreseeable what behavior will be learned for a certain reward function. For example, in a navigation task where the robot approaches a human, the reward function might have a component punishing fast movements. This should ensure that the robot is not learning to behave erratic or to approach humans 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. Often, the reward function needs to be iteratively adapted while learning an appropriate behavior (Fig. 5.1). With standard DRL the task would have to be learned from scratch for each new reward function costing a lot of time.

We investigated Transfer & Meta Learning techniques as a solution to these problems. They transfer knowledge from already learned source tasks to target tasks [27, 57, 72] to improve learning performance on the target task. This enhancement can improve learning speed, resulting in a higher initial performance, or a higher final performance. Our goal is to reduce the cost of learning for tasks where the reward function is changing due to incremental adaptations by the user (Fig. 5.1). The next two sections will introduce our methods and present our results.

5.1.2 Successor Feature Representations

(Contributor: INRIA)

We investigated the application of the Successor Representation (SR) and Successor Feature (SF) framework, two prominent RL Transfer methods, for SPRING's social robotics tasks. Both have restrictions making them not directly applicable. We developed an extension, the Successor Feature Representation (SFR) framework, to overcome these

restrictions. The next section gives a brief introduction to SR and SF before laying out SFR and showing preliminary experimental results.

Background: Successor Representation and Successor Features

The SR framework [12] provides a generalization of value-based RL. It decouples the dynamics of a policy, i.e. which states are visited, from the rewards that are received. The SR function M defines the discounted cumulative future probability of reaching a state s' from state s:

$$M^{\pi_i}(s_t, a_t, s') = \sum_{k=0}^{\infty} \gamma^k \Pr(s' = s_{t+1+k}|\pi) \tag{5.3}$$

where $\Pr(s' = s_t|\pi)$ is the probability to be in state s' at time point t if following policy π. Given this formulation it is possible to define the Q-function of a policy π_i for a reward function R:

$$Q^{\pi_i}(s_t, a_t; R) = \sum_{s' \in S} M^{\pi_i}(s_t, a_t, s')R(s') \tag{5.4}$$

Based on this formulation it is possible to transfer knowledge from solved source tasks, for which policies and SR's (M-functions) have been learned, to a target task by reevaluating the learned policies on the reward function of the target task. This can be done by the General Policy Iteration (GPI) procedure [4]. It defines that for a new reward function R_j an action is taken by evaluating all previously learned policies and the current policy for the new task R:

$$\pi(s_t) = \operatorname*{argmax}_a \max_i Q^{\pi_i}(s_t, a; R) \tag{5.5}$$

GPI selects the action that is optimal according to the best performing policy from previous tasks or the current task. This allows to improve the performance on a new task. In this case the initial policy for this task will be random. Nonetheless, some previous learned policies might not be optimal for this new task but behave better than a random behavior. The SR and GPI procedures allow searching for the action that performs best over all previous learned policies. As the policy for the new task learns and might improve over the previous policies, it will be mostly selected by the GPI procedure at the end of training.

Nonetheless, SR has the problem that its current implementations are only able to handle finite and discrete state spaces, i.e. where $S \in \{s_1, \ldots, s_N\}$. Learning the SR representation M (5.3) for high dimensional state spaces such as images or auditory signals is difficult and not achieved so far. Moreover, for continuous state spaces the computation of the Q-values (5.4) becomes an intractable integral over the continuous state space.

The SF framework [4, 15] extends SR to handle high-dimensional, continuous state spaces $S \in \mathbb{R}^m$. It is based on the assumption that rewards are linearly defined (or approximated) based on state features $\phi(s) \in \mathbb{R}^n$ and a reward weight vector $\mathbf{w} \in \mathbb{R}^n$:

$$R(s_t) = \phi(s_t)^\top \mathbf{w} \tag{5.6}$$

State features $\Phi \subseteq \mathbb{R}^n$ are a low dimensional description of important properties of a state that can be used to define the rewards. For example, an agent that collects different types of foods, it could be a one-hot vector that defines which food type is collected: $\phi(s) = [0, 0]$ if no food is collected, $\phi(s) = [1, 0]$ if apples are collected, and $\phi(s) = [0, 1]$ if berries are collected. The weight vector $\mathbf{w} \in \mathbb{R}^2$ describes then how much each type of food is rewarded. A weight of $\mathbf{w} = [1.0, 2.0]$ gives a reward of $r = 1.0$ for collecting apples and $r = 2.0$ for berries. Based on the linear assumption (5.6) the Q-value can be again decoupled in a part that describes the dynamics of a policy, i.e. the discounted future cumulative features $\psi^{\pi_i}(s_t, a_t)$, and the rewards, i.e. the weight $\mathbf{w}$:

$$\begin{aligned} Q^{\pi_i}(s_t, a_t; \mathbf{w}) &= \mathrm{E}\left\{r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \ldots\right\} \\ &= \mathrm{E}\left\{\phi_t^\top \mathbf{w} + \gamma^1 \phi_{t+1}^\top \mathbf{w} + \gamma^2 \phi_{t+2}^\top \mathbf{w} + \ldots\right\} \\ &= \mathrm{E}\left\{\sum_{k=0}^{\infty} \gamma^k \phi_{t+k}\right\}^\top \mathbf{w} \equiv \psi^{\pi_i}(s_t, a_t)^\top \mathbf{w} \end{aligned} \tag{5.7}$$

The ψ function can be learned with a temporal difference update similar to Q-learning. Given a transition (s_t, a_t, r_t, s_{t+1}):

$$\psi^{\pi_i}(s_t, a_t) \leftarrow \psi^{\pi_i}(s_t, a_t) + \alpha\left(\phi(s_{t+1}) + \gamma\psi^{\pi_i}(s_{t+1}, a_{t+1}) - \psi^{\pi_i}(s_t, a_t)\right) \text{ with } a_{t+1} = \operatorname*{argmax}_a Q^{\pi_i}(s_{t+1}, a; \mathbf{w}_i) \tag{5.8}$$

Transfer from source task to a new target task, i.e. a new reward weight vector $\mathbf{w}$, is performed using the GPI procedure (5.5).

The SF framework is able to handle continuous high-dimensional state spaces with low-dimensional feature spaces [4]. Nonetheless, its assumption on a linear combination of features and a reward weight vector restrict is application. For example, in social robotics we might have as a feature the speed of the robot $\phi_{speed} \in \mathbb{R}$ and as reward function a Gaussian: $r(\phi_{speed}) = \exp(\frac{(\phi_{speed}-\mu)^2}{\sigma})$, where $\mu \in \mathbb{R}$ represents the preferred speed. Such a reward function cannot be represented with a linear combination of features and a weight: $r(\phi_{speed}) \neq \phi_{speed}^{\top}\mathbf{w}$.

Method: Successor Feature Representation

We proposed the novel Successor Feature Representation (SFR) framework [40] to overcome the shortcomings of SR and SF. SFR lends from the SF framework the idea of state features $\phi(s_t) \in \mathbb{R}^n$ that describe the important properties of a state to define the reward function. In difference to SF, the SFR framework does not assume a linear composition of rewards into features and a weight vector but allows general functions: $R(s_t) = f(\phi(s_t))$. From the SR framework, SFR borrows the concept of the expected future cumulative visitation probability (5.3), but instead of being defined over discrete states it is over the continuous features:

$$\xi^{\pi}(s_t, a_t, \phi) = \sum_{k=0}^{\infty} \gamma^k \Pr(\phi_{t+k} = \phi|\pi) \,, \tag{5.9}$$

where $\Pr(\phi_t = \phi|\pi)$ is the probability of observing feature ϕ at time point t following policy π. Similar to the SR framework, the Q-function can be defined using the ξ-function:

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= \sum_{k=0}^{\infty} \gamma^k \mathrm{E}_{p(\phi_{t+k}|s_t,a_t;\pi)}\left\{R(\phi_{t+k})\right\} = \sum_{k=0}^{\infty} \gamma^k \int_{\Phi} \Pr(\phi_{t+k} = \phi|\pi) R(\phi) \mathrm{d}\phi \\ &= \int_{\Phi} R(\phi) \sum_{k=0}^{\infty} \gamma^k \Pr(\phi_{t+k} = \phi|\pi) \mathrm{d}\phi = \int_{\Phi} R(\phi) \xi^{\pi}(s_t, a_t, \phi) \mathrm{d}\phi \end{aligned} \tag{5.10}$$

Given a transition $(s_t, a_t, s_{t+1}, \phi_t)$ the ξ-learning update operator is defined as:

$$\xi^{\pi}(s_t, a_t, \phi) \leftarrow \xi^{\pi}(s_t, a_t, \phi) + \alpha\Big(\Pr(\phi_t = \phi|s_t, a_t) + \gamma \xi_k^{\pi}(s_{t+1}, \bar{a}_{t+1}, \phi) - \xi^{\pi}(s_t, a_t, \phi)\Big) \tag{5.11}$$

where $\bar{a}_{t+1} = \mathrm{argmax}_a \int_{\Phi} R(\phi)\xi^{\pi}(s_{t+1}, a, \phi)\mathrm{d}\phi$.

For transfer, SFR uses the GPI procedure (5.5) similar to SR and SF. Besides the introduction of SFR, we also proved its convergence to the true Q-function and provide a bound on its performance during transfer [40].

The computation Q-values for SFR in (5.10) requires an integral over the features. This is theoretically intractable but can be numerically approximated by introducing a discretization of the feature space. We further make the assumption that reward functions are a sum over independent features:

$$R(\phi) = \sum_{k=1}^{n} r_k(\phi_k) \tag{5.12}$$

where the reward component functions $r_i : \mathbb{R} \mapsto \mathbb{R}$ can be of any form such as Gaussians. As a result the Q-value can be computed as:

$$Q^{\pi}(s, a) = \int_{\Phi} R(\phi)\xi^{\pi}(s, a, \phi)\mathrm{d}\phi = \sum_{k} \int_{\Phi_k} r_k(\phi_k)\xi_k^{\pi}(s, a, \phi_k)\mathrm{d}\phi_k \tag{5.13}$$

where Φ_k is the feature space for each feature dimension and ξ_k^{π} is a ξ-function for the feature dimension k. Moreover, we introduce a discretization of the ξ-function that discretizes the space of each feature dimension k in $U = 11$ bins with the centers:

$$X_k = \left\{\phi_k^{\min} + j\Delta\phi_k : 0 < j < U\right\} \;\text{ with }\; \Delta\phi_k := \frac{\phi_k^{max} - \phi_k^{min}}{U - 1}$$

where $\Delta\phi_k$ is the distance between the centers, and ϕ_k^{min} is the lowest center, and ϕ_k^{max} the largest center. Given this discretization and the decomposition of the Q-function according to (5.13), the Q-values can be computed by:

$$Q^{\pi}(s, a) = \sum_{k} \sum_{x \in X_k} R(x)\xi^{\pi}(s, a, x) \,. \tag{5.14}$$

This formulation removes the integral from (5.10) making the computation of the Q-function feasible without introducing while allowing a general form of reward functions (5.12).

The ξ-function $\xi(s, a, \phi) \in \mathbb{R}$ is approximated with a neural network $\tilde{\xi}$. It takes as input the state $s \in R^m$. For current tasks, the agent has only a low number of discrete actions ($|A| = 3$). A separate neural network is trained for each of the actions. The output of each network is the ξ-value for each of the n feature dimension and each of its U discretized bins.

The ξ-function is updated according to the following procedure. Instead of providing a discrete learning signal to the model, we encode the observed feature using continuous activation functions around each bin center. Given the j'th bin center of dimension k, $x_{k,j}$, its value is encoded to be 1.0 if the feature value of this dimension aligns with the center ($\phi_k = x_{k,j}$). Otherwise, the encoding for the bin decreases linearily based on the distance between the bin center and the value ($|x_{k,j} - \phi_k|$) and reaches 0 if the value is equal to a neighboring bin center, i.e. has a distance $\geq \Delta\phi_k$. We represent this encoding for each feature dimension k by $\mathbf{u}_k \in (0, 1)^U$ with:

$$\forall_{k \in \{1,2,3\}} : \forall_{0<j<U} : u_{k,j} = \max\left(0, \frac{(1 - |x_{k,j} - \phi_k|)}{\Delta\phi_k}\right) \tag{5.15}$$

To learn $\tilde{\xi}$ we update the parameters θ^ξ using stochastic gradient descent following the gradients $\nabla_{\theta^\xi} \mathcal{L}_\xi(\theta^\xi)$ of the loss based on the ξ-learning update (5.11):

$$\begin{aligned} \forall \phi \in \Phi : \; \mathcal{L}_\xi(\theta^\xi) = \mathrm{E}\left\{ \frac{1}{n} \sum_{k=1}^{3} \left(\mathbf{u}_k + \gamma \tilde{\xi}_k(s_{t+1}, \bar{a}_{t+1}; \bar{\theta}^\xi) - \tilde{\xi}_k(s_t, a_t; \theta^\xi) \right)^2 \right\} \\ \text{with } \; \bar{a}_{t+1} = \underset{a}{\operatorname{argmax}} \sum_k \sum_{x \in X_k} R(x) \tilde{\xi}(s_{t+1}, a, \phi; \bar{\theta}^\xi) \end{aligned} \tag{5.16}$$

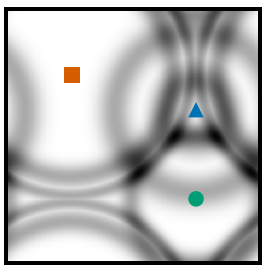
where n is the number of feature dimensions and $\tilde{\xi}_k$ is the vector of the U discretized ξ-values for dimension k.

Experimental Results

The SFR framework has been evaluated in four environments. Three environments with discrete features ($\phi \in \{0, 1\}^N$) and one environment with continuous features ($\phi \in \mathbb{R}^N$). We summarize here the results for the environment with continuous features, the racer environment (Fig. 5.2), as continuous features are usually prevalent in robotic tasks, such as, distances to humans or objects. Please refer to [40] for further details about the racer environment and the results for the other environments.

In the racer environment, the agent starts at a random position on a 2D map. It drives around the map for 200 timesteps before the episode ends. Similar to a car, the agent has an orientation and momentum, so that it can drive straight, or in a right or left curve. The agent reappears on the opposite side of the map if it exits one side. The distance to three markers are provided as features $\phi \in \mathbb{R}^3$. Rewards depend on the distances $r = \sum_{k=1}^{3} r_k \phi_k$, where each component r_k has one or two preferred distances defined by Gaussians. We evaluated the agent on 37 sequentially learned tasks by using the GPI transfer procedure between tasks. For each tasks, a random reward function is generated where the number of Gaussians and their properties (μ, σ) are randomly sampled for each feature dimension. Fig. 5.2 (a) shows a reward function where dark areas depicting higher rewards. The state space is a high-dimensional vector $s \in \mathbb{R}^{120}$ encoding the agent's position and orientation. The 2D position is encoded by a 10×10 grid of two-dimensional Gaussian radial basis functions. Similarly, the orientation is also encoded using 20 Gaussian radial basis functions.

(a) Racer Environment



(b) Average return per task over environment steps

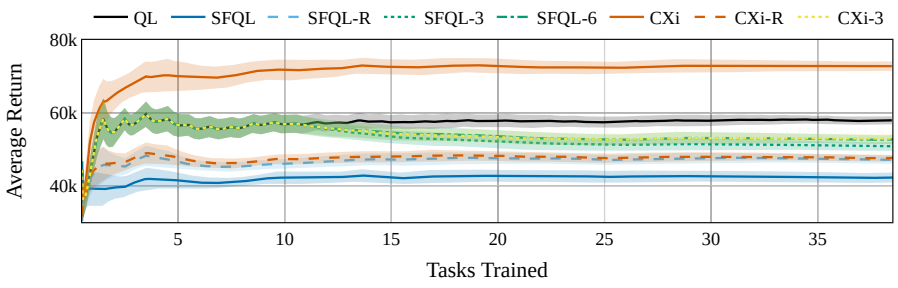


Figure 5.2: (a) Example of a reward function for the racer environment based on distances to its 3 markers. (b) SFR-learning reaches the highest average reward per task. SF-QL yields a performance even below QL as it is not able to approximate correctly the reward function with its linear combination of weights and features. The average over 10 runs per agent and the standard error of the mean are depicted. TODO: only plot QL, SFQL and SFRQL

Three agents were evaluated: 1) Standard Q-Learning (QL) that uses no transfer of knowledge and learns every task from scratch. The Q-function is approximated by a fully connected feedforward network with bias and a ReLU activation for its hidden layers. It has 2 hidden layers with 20 neurons each. 2) The SF (SF-QL) procedure that assumes a linear decomposition of rewards. As the reward functions are not able to be represented as a linear decomposition of a weight vector $\mathbf{w}$ and the features, an approximated weight vector $\tilde{\mathbf{w}}$ was learned that minimizes the mean squared error to the true reward function: $\phi^\top \tilde{\mathbf{w}} \approx R(\phi)$. SF-QL uses a feedforward network with bias and a ReLU activation for hidden layers to approximated the ψ-function. It has for each action a separate fully connected subnetwork. Each subnetwork has 2 hidden layers with 20 neurons each. 3) The SFR (SFR-QL) procedure which also uses a feedforward network with 2 hidden layers and 20 neurons each to approximate the ξ-function for each action.

We evaluated each agent on 37 tasks for 10 runs per agent and compared their average return per task over the environment interaction steps (Fig. 5.2 - b). SFR-QL reached the highest performance of all agents outperforming QL and SF-QL. SFQL reached only a performance below QL. SFQL is not able to approximate sufficiently well the Q-function as their reward functions depend on a linear reward model which can not represent well the general reward functions in this environment.

Conclusion

We developed SFR, an extension to the Transfer Learning frameworks of SR and SF. SFR allows to handle continuous and high-dimensional state space in comparison to SR, and general reward function in comparison SF. Our preliminary experiments in a simulated environment showed its advantage over the SF framework, and traditional non-transfer RL. In future experiments, we will investigate its application to relevant social robotic tasks for SPRING.

5.1.3 Radial-Basis-Function Layers for Variational Meta Learning

(Contributor: INRIA)

Additionally to the SFR framework, we investigated [3] task conditioned policies methods to reduce the amount of needed data when iteratively searching for an appropriate reward function. Task conditioned policies are a Meta-RL approach able to adapt to new tasks, i.e. reward functions, using only few observations. These methods learn a meta-policy, that is conditioned on a latent vector representation z of a task. How to represent tasks and solve them is learned on a set of source tasks. Given a new target task, a small amount of observations are collected to compute its task representation and to condition the meta-policy to it. The idea is to learn the meta-policy before searching for a specific reward function on the real robot in simulation. The procedure can also be used to adapt the robots behavior quickly to new environments. For example, a social robot might be used to navigate in a care center for elderly or an office. In the care center the robot should avoid getting too close to people to avoid that they feel unsafe. Whereas, in an office environment this restriction might be less important and the robot could move closer around people to reach its goal position faster. Being able to adapt a robot behavior, i.e. the reward function and its resulting behavior, quickly to the specific needs of a certain new environment is essential for the practical application of social robotics.

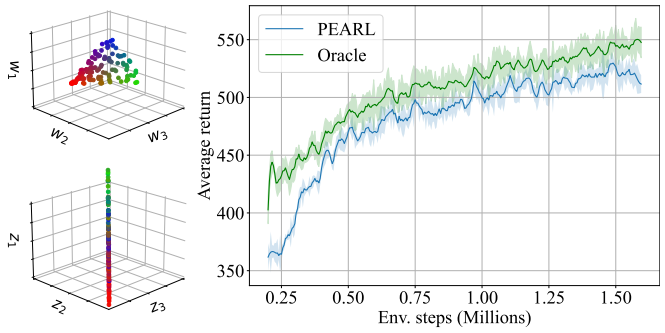


Figure 5.3: Graphic visualization of the posterior collapse of PEARL. (left-top) The ground truth task representation, where each task corresponds to a coloured dot. The three primary colors correspond to the strength of each of three reward component weights. (left-bottom) PEARL's learned task representation uses only one dimension (z_1) and exhibits posterior collapse in the two other dimensions (z_2 and z_3). (right) As a result, PEARL's learning performance compared to a task conditioned policy using the ground truth (Oracle) is reduced. The proposed RBF-PEARL aims to close this gap by mitigating the negative impact of posterior collapse during training.

A promising research direction for task conditioned polices are variational architectures [22], such as PEARL [39]. Instead of learning a deterministic representation of tasks, PEARL learns to represent them via a distribution, exploiting the flexibility of probabilistic models and the representation power of deep neural networks. We investigated the usability of such variational meta-RL frameworks for social robotic tasks [3], and realized that the learned encoder suffers from posterior collapse resulting in a reduced learning performance (Fig. 5.3). We propose to use a radial

basis function (RBF) layer [5] to transform the task representation before giving it to the downstream task conditioned policy, thus constructing an embedding that is more suitable to represent tasks. RBF networks are universal function approximators [33] whose parameters can be learned, and exhibit interesting results in classification tasks [9, 36, 64] as well as in value-learning for continuous action DRL [2]. In the following, we first discuss work related to our topic and methods, then introduce our methodology, to finally present experimental results for two simulated social robotics tasks.

Background

In this work, we would like an agent to adapt quickly to a new reward function using only a few observations. We formalize this problem as a meta-RL problem. Each reward function represents a task $\tau \in \mathcal{T}$ with $\tau = \{\mathcal{R}(s_t, a_t, s_{t+1})\}$. We assume a distribution $\rho(\tau)$ over the task. We differentiate two meta-learning phases: 1) meta-training and 2) meta-testing. During meta-training, a number of training tasks is sampled according to $\rho(\tau)$. Based on these training tasks, a task conditioned policy $\pi(a|s, \tau)$ is learned. During meta-testing, a different set of test tasks is sampled from $\rho(\tau)$ and the adaptation of the learned policy to these tasks is measured.

We chose PEARL [39] as our meta-RL algorithm. PEARL learns a task-conditioned policy $\pi(a|s, z_\tau)$ where $z_\tau \in Z = \mathbb{R}^d$ is a low-dimensional task representation. The task representation z_τ conditions the policy towards maximizing the return of the reward function for task τ. The representation $z_\tau = f(c^\tau)$ is computed based on observed transitions from a task, called the context $c^\tau = \{c_n^\tau\}$, where $c_n^\tau = (s_n, a_n, r_n, s'_n)$. PEARL uses a variational method, similar to variational autoencoders (VAEs) [22], to estimate the posterior distribution of the low-dimensionsal representation given the context $p(z|c)$. It approximates the posterior with an inference network $q_\phi(z|c)$ parametrized by ϕ. The network is trained on a log-likelihood objective resulting in the following variational lower bound:

$$\mathbb{E}_\tau \left[\mathbb{E}_{z\sim q_\phi(z|c^\tau)} \left[\mathcal{R}(\tau, z) + \beta D_{KL}(q_\phi(z|c^\tau)) \| p(z)) \right] \right] \tag{5.17}$$

where $\mathcal{R}(\tau, z)$ is the return for task τ using the policy conditioned on z and $p(z)$ is a standard Gaussian prior over z. The inference network is a product of independent Gaussian factors for each transition in c^τ:

$$q_\phi(z|c^\tau) \propto \prod_{n=1}^{N} \mathcal{N}\left(f_\phi^\mu(c_n^\tau), f_\phi^\sigma(c_n^\tau) \right) \tag{5.18}$$

where f_ϕ^μ and f_ϕ^σ are represented by a neural network.

During meta-training, the policy $\pi(a|s, z)$ is learned using the soft actor-critic (SAC) [19] algorithm. SAC is off-policy, consisting of an actor $\pi_{\theta_\pi}(a|s, z)$ and a critic $Q_{\theta_Q}(s, a, z)$ network. PEARL jointly trains the inference, actor and critic networks using the reparameterization trick similar to VAEs [22]. During a meta-training step, the training procedure has two phases: 1) data collection and 2) network parameters update. In the data collection step, a replay buffer $\mathcal{B}^\tau$ is filled with the transitions from K trajectories for each training task τ. Then, for each trajectory PEARL samples a task representation $z \sim q_\phi(z|c^\tau)$ to condition the policy where c^τ is sampled from the replay buffer $\mathcal{B}^\tau$. During the second phase, the procedure updates the network parameters for each training task τ. It first samples a batch of context c^τ from recently sampled transitions in the replay buffer $\mathcal{B}^\tau$. Then, the task representation $z \sim q_\phi(z|c^\tau)$ is sampled from the posterior distribution of z given the context c^τ. The critic and actor networks are updated using the task representation and independently sampled transitions from the whole replay buffer. The loss of the critic is given by:

$$\mathcal{L}_{critic} = \mathrm{E}_{\substack{(s,a,r,s')\sim\mathcal{B} \\ z\sim q_\phi(z|c)}} \left[Q_\theta(s, a, z) - (r + \dot{V}(s', \dot{z})) \right]^2 \tag{5.19}$$

where $\dot{V}$ is the value, i.e. the maximum Q-value, of a target network, and $\dot{z}$ indicates that gradients are not being computed through it. A target network is necessary here because directly implementing Q learning with neural networks proved to be unstable in many environments [32]. The actor loss is given by:

$$\mathcal{L}_{act} = \mathrm{E}_{\substack{s\sim\mathcal{B}, a\sim\pi_\theta \\ z\sim q_\phi(z|c)}} \left[log(\pi_\theta(a|s, \dot{z}) - (Q_\theta(s, a, \dot{z})) \right] \tag{5.20}$$

The loss of the inference network for the task representation is composed of the critic loss and the Kullback–Leibler divergence term from (5.17):

$$\mathcal{L}_\phi = \mathrm{E}_{\substack{(s,a,r,s')\sim\mathcal{B} \\ z\sim q_\phi(z|c)}} \left[\mathcal{L}_{critic} + \beta D_{KL}(q_\phi(z|c^\tau)) \| p(z)) \right]. \tag{5.21}$$

For meta-testing, a test task τ is first explored for a few hundred time steps. The policy $\pi_e(a|s, z_e)$ used for exploration is conditioned on a task representation sampled from the Gaussian prior $z_e \sim p(z)$. After the exploration phase, the context c^τ collected with π_e is used to compute the final task representation given by the sample mean of the posterior mean: $z^\tau = \frac{1}{N} \sum_{n=1}^{N} f_\phi^\mu(c_n^\tau)$. More details on the meta-testing phase are provided in the experimental section.

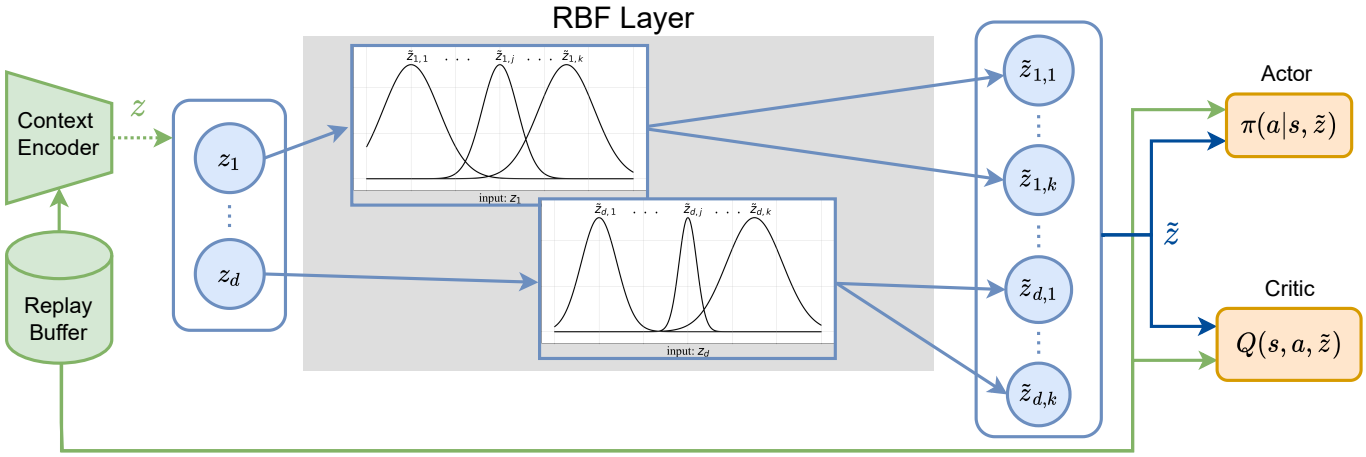


Figure 5.4: **PEARL-RBF Architecture.** The proposed meta-RL procedure learns to adapt to unseen test reward functions. The context encoder uses data from the replay buffer to infer the posterior over the latent context variable z. The latent context sampled from the posterior $z = (z_1, \dots z_d)$ is then fed to the RBF network, which uplifts every input dimension m to a different k-dimensional representation: $z_m \rightarrow (\tilde{z}_{m,1}, \dots, \tilde{z}_{m,k})$. The resulting task representation $\tilde{z}$ is used to condition the actor and critic network.

Method: RBF-PEARL

We noticed that in several scenarios PEARL suffers from posterior collapse of the learned task representation z. As a result, not all dimensions of z are used. The task information is compressed in few dimensions making it difficult for the downstream policy and critic network to learn from z. To compensate for this, we propose to transform the task representation $\tilde{z} = \varphi(z)$ to a representation that is easier to process for downstream networks. Inspired by [2, 9], we propose the usage of radial basis function (RBF) layers to transform the task representation.

We construct the RBF layer (Fig. 5.4) based on the idea of RBF networks [5] which are universal function approximators [33]. RBF networks consist of a layer of hidden neurons that have a Gaussian activation function. The output of the networks is a weighted sum over the Gaussian activations. In a similar manner, our proposed RBF layer consist of a layer of neurons. For each input dimension $z_j \in \mathbb{R}$ from the original task representation exist N RBF neurons: $\tilde{z}_{j,1}, \dots, \tilde{z}_{j,N}$. Each neuron represents a radial basis function having a Gaussian shape:

$$\tilde{z}_{i,j} = \exp\left(-\delta_{i,j}||z_i - c_{i,j}||^2\right), \tag{5.22}$$

where $\delta_{i,j} \in \mathbb{R}$ is a scaling factor and $c_{i,j} \in \mathbb{R}$ is the center, i.e. the point of the highest activation. δ and c are the parameters of the RBF layer which can be either fixed or trained using gradient descent based on the loss function of downstream networks.

In summary, we propose to transform the task representation using a RBF layer: $\tilde{z} = \varphi(z)$. The resulting representation is then given to the task conditioned actor $\pi_\theta(a|s, \tilde{z})$ and critic $Q(s, a, \tilde{z})$ network.

Experimental Results

We evaluated our approach on three different environments of which we report here results on two that are inspired from social interaction situations. See [3] for the results of all environments.

In the first environment, the agent, represented by a robotic head, needs to learn how to control its position following different criteria (e.g. maximizing the number of people in the field of view or facing the speaker). In the second environment, the agent must learn to navigate safely and in a socially-compliant manner through a crowd of people. First we want to show the effectiveness of our proposed methodology to generate different behaviors in human robot interaction tasks. Secondly, we assess whether or not the use of RBF layers is beneficial for variational meta-RL in terms of training and adaptation efficiency.

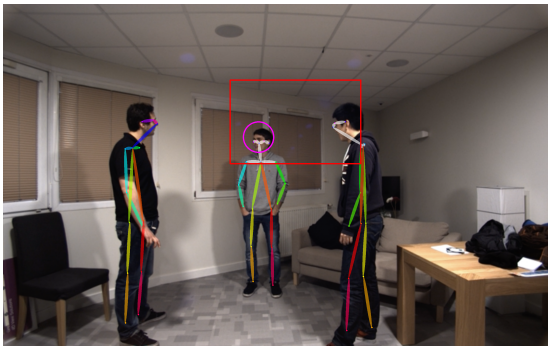
Baseline The most natural baseline to the proposed RBF-PEARL is standard PEARL. However, directly comparing with PEARL seems unfair, since adding RBF layers increases the number of parameters of the actor and critic networks. We therefore adjust the number of parameters of the actor and critic networks of the PEARL baseline so as to match the number of parameters of the RBF-PEARL. Beyond this adjustment, both methods use separated actor and critic networks, and both networks consist in a 3-layers MLPs with 300 neurons per layer. Both methods use a

3-layer MLP with 200 neurons per layer as encoder. In more details, if the latent dimension is d, and we use one RBF layer with k neurons, the first layer of the actor/critic networks would have $300d$ and $300kd$ parameters for PEARL and RBF-PEARL, respectively. For the sake of a fair comparison, we add an extra layer at the beginning of PEARL's actor/critic network with kd output neurons, thus yielding the number of parameters of PEARL and RBF-PEARL actor and critic networks comparable. Unless otherwise stated, we use $k = 9$.
As a separate baseline, we used a modified version of the soft actor critic algorithm. We trained one agent per tasks, using only 200 observations. However we strongly increased the numbers of gradient step performed per observations in order to force the network to learn a behavior with only this 200 observations. We used this baseline to compare the performances classical RL algorithms (here SAC) trained on 200 observation can reach with respect to RBF-PEARL and PEARL with 200 adaption steps. We except the performances of these agents trained with soft actor critic algorithm to be lower than the performances of PEARL and RBF-PEARL.

Evaluation protocol For meta-training, we sample 100 tasks for the three environments. Evaluation is performed on 20 meta-testing tasks that are different from those used at meta-training time. To evaluate on each meta-testing task τ, we collect 200 time steps with a task representation z_e sampled from the standard Gaussian prior. Then we aggregate this 200 time steps to compute a task representation z^τ sampled from the posterior estimation given by the encoder. To compute the final test-time performance, we record the performance of the policy associated to the task representation z^τ for one episode. We repeat the training process 5 times, and report the mean performance and the associated standard error.

Gaze control Environment Inspired by recent research in RL for social robotics [26] to design this environment, where we aim to learn a gaze control strategy for a robotic head. In our setup, the robot observations are multimodal consisting of visual, auditory, and proprioception cues. To this end, we assume normalized scene coordinates of size 2×1. Regarding the visual observations, we assume a regular head camera that extracts pose cues from the 0.4×0.3 field of view (FoV). In more detail, we assume that a multi-person pose estimation method is available, and provides one heatmap for each of the $J = 18$ landmarks (e.g. nose, neck, left shoulder, right hip). The heatmap associated to each landmark indicates the probability of the presence of that landmark at every position. Since state-of-the art pose estimators provide these heatmaps in low resolution, the visual observations will consist on J 7×7 heatmaps. Regarding the auditory observations, we emulate the output of a sound source localization algorithm. Given the precision of current sound source localization methods, it seems reasonable to represent the auditory input as a 14×8 heatmap corresponding to the entire scene. Each of its cells corresponds to the probability of having an active speaker in this direction. Importantly, while the visual features correspond to the current field of view of the camera, the auditory features correspond to the entire scene, since audio localization is not limited by the camera's FoV. Lastly, the propri-

(a) Gaze control task



(b) Results of the Meta-Adaptation

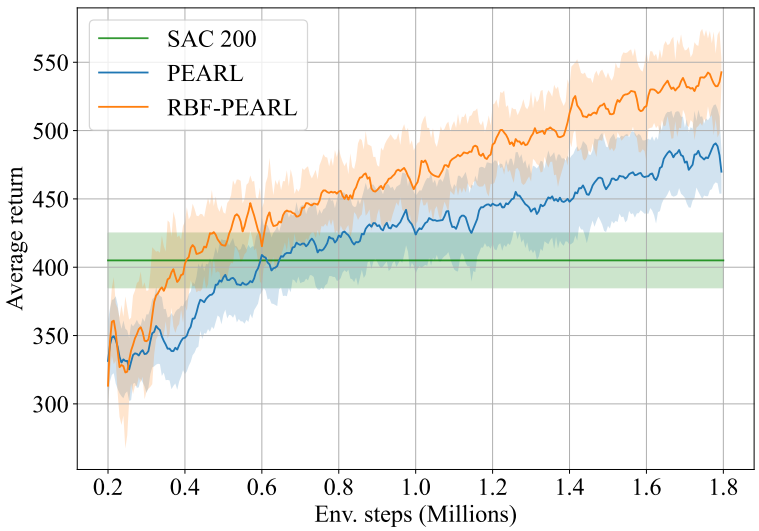


Figure 5.5: (a) **Illustration of the gaze control environment.** The field of view is shown in red, the active speaker in purple, and the visible/non-visible landmarks are shown in white/color respectively. (b) RBF-PEARL also outperforms PEARL in the test-task performance vs. samples collected during meta-training on the non-linear family of reward functions on the gaze control environment.

oception cues consist of the coordinates of the robots current field of view center. It is encoded using a $\mathbb{R}^2$ vector representing both the pan and tilt angles of the robotic head, defining to which part of the scene corresponds the visual input. An observation of the gaze control environment is shown in Figure 5.5 (left).

Regarding the action space of the agent A, we set it to $[-1, 1]^2$,corresponding to the pan and tilt angular velocities, respectively. We choose to normalize the action space so as to stabilize the network learning. The maximum pan and tilt velocities correspond to shifting the camera field of view by 0.16 and 0.11 in normalized scene coordinates at every time step respectively.

We define three reward components that will be combined to generate various tasks (i.e. reward functions). We will focus on the number of people in the field of view, the presence of a speaker in the field of view, and reducing spurious robot movements. These three components are very generic, and relevant for any social robot. The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 3$.

We want to reward the robot for having people in the field of view of the camera, since this means the robot would be looking at people. Naively reward the number of people within the field of view leads to an action policy that explores until finding a person, and then follows the person. A more natural behavior is to check on previously detected people. We therefore propose to use a visual reward component that depends on the last time the agent saw a person. More precisely:

$$R_{\text{vis}} = \sum_{p \in P_{\text{vis}}} 2 - \exp(-t_p) \tag{5.23}$$

where P_{vis} is the set of people whose face is in the visual field of view and $t_p \in [0, \infty]$ is the time since the person was last seen (before the current frame). When a person keeps on being in the field of view, the reward is close to 1. When a person not seen for long time reappears in the field-of-view, the reward is close to 2. In this way, the desired behavior is encouraged.

Regarding audio, we would like the robot to look at the speaking person(s), and therefore we define the audio component of the reward function as:

$$R_{\text{aud}} = \begin{cases} 0 & \text{Nobody speaks} \\ -0.5 & \text{Speakers are outside the FoV} \\ 2|P_{\text{aud}}| & \text{Speakers within the FoV} \end{cases} \tag{5.24}$$

where P_{aud} is the set of speakers in the field of view.

Finally, we would like to penalize large and fast movements since they are quite unnatural in social interactions. This is why we propose a movement (negative) component of the reward function, defined as:

$$R_{\text{mov}} = -K_{\text{mov}} \sqrt{a_{pan}^2 + a_{tilt}^2} \tag{5.25}$$

where a_{pan} and a_{tilt} are the two components of the action space, and $K_{\text{mov}} = 16$ is a constant to put R_{mov} in a similar numeric range as R_{vid} and R_{aud}.

In order to construct different tasks (reward functions) we propose to use the defined components in two different ways. First, with simple combinations as in [26], and then with more complex ones (i.e. non-linear). The first family of reward functions is generated by sampling convex combinations of the three components defined above:

$$R^\tau = \omega_{\text{vis}}^\tau R_{\text{vis}} + \omega_{\text{aud}}^\tau R_{\text{aud}} + \omega_{\text{mov}}^\tau R_{\text{mov}} \ , \tag{5.26}$$

where $\omega_{\text{vis}}^\tau, \omega_{\text{aud}}^\tau, \omega_{\text{mov}}^\tau \in [0, 1]$ are random convex weights, meaning that $\omega_{\text{vis}}^\tau + \omega_{\text{aud}}^\tau + \omega_{\text{mov}}^\tau = 1$.

We also wanted to compare the algorithms in more complex environments. To that aim, we design our second family of reward function using random multi-layer-perceptron (MLP) networks. These input the value of the three reward components defined above. The MLPs have 1 to 3 layers with 4 to 6 neurons each, that are activated with a sigmoid with probability 0.75. The number of layers, the neurons per layer, the activation and the weights are sampled randomly. Formally, we write:

$$R^\tau = f^\tau(R_{vis}, R_{aud}, R_{mov}; W^\tau) \tag{5.27}$$

where f^τ represents the sampled MLP network with sampled connection weights W^τ.

We report the average return over the set of meta-testing tasks over the met-training iterations (Fig. 5.5 - b). More precisely we plot the average return mean and standard deviation over the five independent runs, for the gaze control environment with convex (top) and non-linear (bottom) combinations of reward components.

Generally speaking, both PEARL and RBF-PEARL are able to provide a better adaptation starting point with the training progress. In addition, we observe that RBF-PEARL has a steeper learning curve than PEARL on both types of reward functions. More precisely for convex combinations of reward components, RBF-PEARL performs comparably to PEARL during the first 400k steps. From this point on, RBF-PEARL systematically outperforms PEARL by a margin of

20-30. For the non-linear combinations of reward components, RBF-PEARL exhibits superior performance from 600k meta-training steps on, by a margin of roughly 50. Overall, on these two families of tasks RBF-PEARL is faster than PEARL and has better asymptotic performance, thus show-casing the benefit of using the RBF layer. As excepted, both PEARL and RBF-PEARL achieves better performances than the batch of agents trained using the soft actor critic algorithm with only 200 observations (SAC 200). In the Linear Gaze Control environment, the average performances reach by agents trained with the soft actor critic algorithm is inferior to the performances of PEARL and RBF-PEARL by a margin of 100. In the Non-Linear Gaze Control environment, the gap is lower between PEARL and SAC 200 as PEARL outperforms SAC 200 by a margin of 50. With RBF-PEARL, the gap of performances with SAC 200 is more significant as RBF-PEARL exhibits superior performance by a margin of 130.

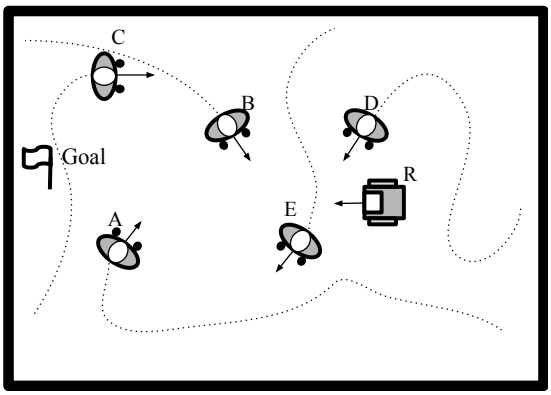
Social navigation environment For this task, we took inspiration from social group and crowd navigation tasks [10, 71]. We propose to learn a navigation strategy for a mobile robot. Our simulation environment is an empty room of dimension 15×10 m. The room is populated with five human agents, whose position at the beginning of each episode is randomly initialized, see Figure 5.6. Likewise, we randomly sample a robot goal position at the beginning of an episode. The robot should reach the goal position before the end of the episode without disturbing the human agents. In our setting, each human agent is given a random goal position. During an episode the human agent will go towards the goal position and will be assigned a new one after reaching the original goal. To simulate the behavior of human agents, we model the human agents' motion using a social force model [35] to generate plausible trajectories. This framework also limits the amount of collision between human agents. In our setup, the robot has access to the coordinates of all people in the scene, their velocities and their orientations. The robot also has access to its own velocity, coordinates and goal position. While the robot always begins the episode at the same position (coordinates $(14, 5)$), its goal position is randomly sampled following $x \sim \mathcal{U}(0.0, 2.0)$ and $y \sim \mathcal{U}(0.0, 10.0)$, for the x and y coordinates respectively. Lastly, the action space is a continuous two dimensional space set to $[-15, 15] \times [-2, 2]$, which corresponds respectively to the angular velocity and linear velocity. The maximum value of the linear and angular velocity are chosen such that the robot can go as fast as any human agent in the scene. In this environment, we use a smaller number of neurons $k = 5$ for our RBF-layer.

We define a set of five reward components that will be combined to generate various tasks. The dimension of the latent space of both PEARL and RBF-PEARL is set to $d = 5$ for this environment. First, the goal component R_g is designed to reward the agent for reaching the goal position:

$$R_g = 1 - \frac{d(r, g)}{D} \tag{5.28}$$

where $d(r, g)$ is the distance between the robot and the goal and D is a normalizing factor to guarantee that the goal component stays within $[-1, 1]$.

(a) Social navigation task



(b) Results of the Meta-Adaptation

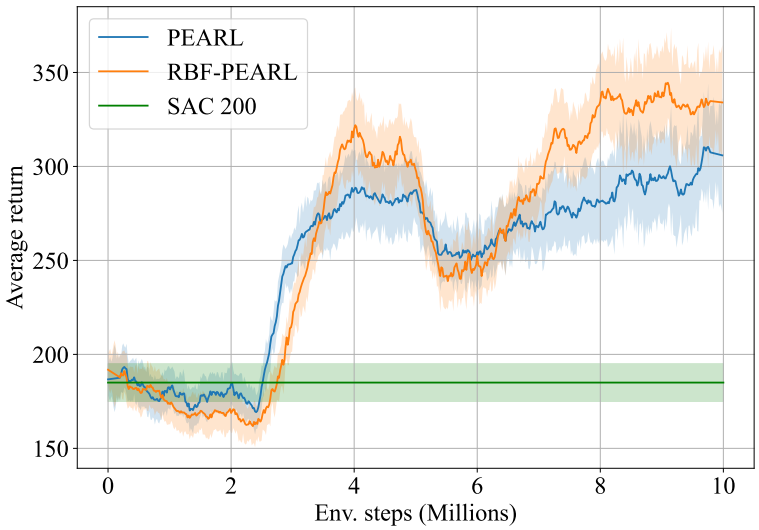


Figure 5.6: **Schematic representation of the social navigation environment**. The robot R must move towards its goal position while navigating around five human agents $(A, \ldots, E)$. **Results on the social navigation environment.** After 6 millions steps, RBF-PEARL outperforms PEARL in the test-task performance vs. samples collected during meta-training on the family of reward functions corresponding to the social navigation environment.

Second, the collision component R_c is built to penalize collisions between the robot and human agents:

$$R_c = \begin{cases} 0 & d(r, h_i) > d_c \\ -1 & d(r, h_i) < d_c \end{cases} \tag{5.29}$$

where $d(r, h_i)$ is the distance between the robot and the human agent i and d_c is the collision threshold between the robot and the human agent.

Third, the social component R_s is designed to reward the robot in maintaining a safe distance from all human agents. The social component depends on the distance between the robot and each of the human agent. If the distance between the robot and one of them is below a certain threshold, the robot will be penalized. The closer the robot is to the person, the higher the penalty will be. If the robot is close to more than one human agent, only the closest person to the robot is taken into consideration (that is to say the human agent that will generate the lowest reward):

$$R_s = \min_i \left[\frac{d(r, h_i)}{d_s} - 1 \right] \tag{5.30}$$

where $d(r, h_i)$ is the distance between the robot and the human agent i and d_s can be understood as a threshold. Indeed, if the minimum distance between the robot and a human is below d_s, the reward becomes negative. Thus, d_s can be seen as the distance at which the robot enters the comfort zone of people.

Fourth, the approach component R_a is designed to reward the robot for positioning itself so as to avoid making other humans agents aware of it. This component is inspired from the literature in social robotics, see [41]. In this paper, the authors tried to quantify how aware people are of a robot when it approach them. They use the relative positions and orientations of the robot and the human agent to evaluate the awareness of the robot by the human. In our approach, we want to minimize the distraction created by the robot when it navigates. Therefore we are trying to minimize the awareness of the robot by the human agents in the scene. We compute the awareness of the robot by each human agent and use this as our approach component by rewarding low awareness of the robot. The approach component is defined for each human agent. It is a combination of visibility and direction. While the visibility assesses how much visible the robot is for the human agent, the direction assesses if the robot is going in a direction that would make the human agent more aware/afraid of it.

The visibility for human agent i is defined as :

$$R_{\text{visible},i} = \begin{cases} 1 - \frac{\theta_{i,r}}{\theta_{th}} & \theta_{i,r} < \theta_{th} \\ -\frac{\theta_{i,r} - \theta_{th}}{\pi - \theta_{th}} & \text{otherwise} \end{cases} \tag{5.31}$$

where θ_{th} is a threshold angle from which the robot is visible to the human agent and $\theta_{i,r}$ is the angle of the robot relative to the human agent i motion. If the robot is in front of the human agent, the value will be close to 1, and if the robot is in the back of the human the value will be close to -1. The direction for human agent i is defined as :

$$R_{\text{direction},i} = \begin{cases} 1 - \frac{\theta_{r,i}}{\pi/2} & \theta_{i,r} < \theta_{th} \\ 1 & \text{otherwise} \end{cases} \tag{5.32}$$

It is designed to address how the human agent perceives the robot coming towards him. If the robot is coming closer to the human agent, he/she will become more aware of it and be distracted by it. On the other hand, even if the robot is visible to the human agent, if it moves away from him/her, it will be less likely that the human agent will be distracted by the robot. To compute the approach reward component, we compute the minimum over the visibility/direction product, over the human agents:

$$R_a = \min_i R_{\text{visible},i} \cdot R_{\text{direction},i} \tag{5.33}$$

Fifth, the velocity component R_v is designed to penalize the robot for going too fast when it is in front of people. It is preferable to avoid having the robot being fast in front of people as they may be disturbed or even afraid by it. Thus we define a velocity component which penalizes the robot for fast-moving if it is visible to other human agents in the scene. The velocity component for each human agent i is defined as:

$$R_{v,i} = \begin{cases} -e^v(1 - \frac{\theta_{i,r}}{\theta_{th}}) & \theta_{i,r} < \theta_{th} \\ 0 & \text{otherwise} \end{cases} \tag{5.34}$$

where θ_{th} is a threshold angle from which the robot is visible and $\theta_{i,r}$ is the angle of the robot relative to the human agent motion. To compute the reward component we take the same methodology as with the social and approach component, that is to say that we choose the minimum value obtained from all the human agents in the scene:

$$R_v = \min_i R_{v,i} \tag{5.35}$$

In order to evaluate the meta-RL algorithms in this setting, we propose to test it with convex combinations, as in the gaze control environment. This social navigation environment is more challenging because the higher number of reward components makes it more difficult for the learning process, and for the behavior generation as there is a wider variety of possible reward functions. Thus our family of reward functions is generated using convex combinations of the five components defined above:

$$R^\tau = \omega_g^\tau R_g + \omega_c^\tau R_c + \omega_s^\tau R_s + \omega_a^\tau R_a + \omega_v^\tau R_v \tag{5.36}$$

where $\omega_g^\tau, \omega_c^\tau, \omega_s^\tau, \omega_a^\tau, \omega_v^\tau \in [0, 1]$ are the randomly sampled convex weights leading to task τ, meaning that $\omega_v^\tau + \omega_c^\tau + \omega_s^\tau + \omega_a^\tau + \omega_v^\tau = 1$.

In Figure 5.6, we report the average return over the meta-testing tasks with the progress of the meta-training for PEARL and RBF-PEARL. As in the previous environment, we report mean and standard deviation over five runs. Similarly to the previous case, the performance of the two methods look similar during the first steps of the training. More importantly, the RBF layer seems to have a positive impact on the asymptotic performance. Indeed, after 6 million steps, the RBF-PEARL algorithm performs better than PEARL. Similarly to the gaze control environment, both PEARL and RBF-PEARL perform better than SAC-200 by a large margin (170).

Conclusion

In this exploratory study, we investigated the use and limitations of variational meta-RL for social robotics. We showed that meta-RL successfully learns to adapt quickly (within 200 steps) to different reward function formulations which can help to identify the reward function that describes a wanted social behavior faster. Nonetheless, state-of-the-art methods exhibited a posterior collapse in our task, which is problematic in meta-RL since the encoder is supposed to accumulate information for better generalization, and collapsed encoding dimensions cannot do so. We started investigating how to mitigate posterior collapse in variational meta-RL by adding a RBF layer after each encoded dimension. Our algorithm improves the performances of meta-RL algorithm for reward design in social robotics. Our RBF layer improves asymptotic performances of the PEARL algorithm in several different environments, all inspired of social robotics tasks. The PEARL algorithm learns a sub-optimal representation of the task. While we do not solve this issue, the RBF-PEARL algorithm mitigates the effect of this sub-optimal representation, providing the actor and critic network with a different representation of the task using the dimensions of the task representation that do not suffer of posterior collapse.

5.2 Co-Speech Gesture Generation

(Contributor: INRIA)

We investigated a neural architecture for the generation of upper body gestures for ARI (movement of arms and head) during conversations. The architecture takes as input the utterances that ARI will speak in text form and generates the movement of ARI's joints. The architecture is based on a Generative Adversarial Network (GAN) [61] model and trained on recorded video data from humans giving talks during TED conferences [62].

Method

Our architecture is based on a Generative Adversarial Network (GAN). GANs are composed of two parts. 1) A Generator network that generates the target output (upper body joint positions) for a given input (utterance as text). 2) A Discriminator network that learns to identify if a given gesture is either artificially generated or a real recorded gesture. The goal of the Generator is to fool the Discriminator that its outputs are real whereas the Discriminator tries to identify the generated data. As a result the Generator learns over time to generate more and more realistic gestures. Our GAN model is based on the architecture by [61]. In difference, our architecture receives as input only the text of the speech, but no audio recordings. Moreover, the text encoder in our architecture is replaced by a pretrained Transformer model based on [58]. Besides these changes we also performed ablation studies introducing Transformer decoder layers that replace the GRU layers of the Discriminator and the Generator in [61]. We differentiate between both versions as the GRU network (gru), Fig. 5.7, and the Transformer network (trans), Fig. 5.8.

Generator Network The generator network takes three inputs per data sample: a) The spoken text as a string. b) The initial poses of the speaker. These represent the joint poses of the previous 4 time frames, so that the goal is to generate poses that continue from these. Poses are represented as directional vectors representing relative positions

of child joints from parent joints. Each pose encodes 9 joints where each joint is represented by a 3D vector ($p \in \mathbb{R}^{27}$). c) The ID ($\in \mathbb{N}$) of the speaker which is used to compute a style embedding to reflect inter-person variability.

Both networks (gru and trans) use a pretrained sentence encoder[1] to process the spoken text. The encoder is based on the Transformer architecture in [58]. It outputs a sequence of sentence embeddings of variable length L where each embedding is a 384 dimensional vector ($f_{sentence} \in \mathbb{R}^{L \times 384}$). For the GRU network the sequence is averaged over its length L to have a fixed size input for its GRU units using a pooling average procedure. The averaged embedding is given at each computational step to the GRU network. The Transformer network takes the full sequence of sentence embeddings as memory input.

The GRU network receives for its first four computational steps the initial pose as an input with an additional bit input $b \in \mathbb{R}$ set to 1 that encodes that the input sequences are initial poses. Afterward, it receives for each generated pose $\hat{p}$ an input pose p where all values are zero. In this case, the bit input is zero. The Transformer network receives all positions (initial ones and zero vectors) and the bit information as vectors.

Finally, both networks receive as input a style embedding f_{style} which is learned by a variational network. It receives the speaker's ID as a one-hot vector. A set of fully connected layers maps the ID to a style embedding space $f_{style} \in \mathbb{R}^{16}$. The network uses a sampling process similar to VAEs to learn the style mapping.

The Transformer network receives as additional input a positional encoding for each of the 34 input poses. The positional encoding follows the standard formulation for Transformers with an additional drop-out layer at the end.

The GRU network consists of 4 bidirectional GRU layers on top of each other, followed by two fully connected layers. The first has the same dimension as the GRU layer's output, and the second a size half of the dimension as the GRU layer's output. We evaluated different sizes for the output of the GRU layers. The Transformer network has 6 layers of Transformer decoder blocks with 8 heads and feedforward network models of dimension 1024. As the GRU network, it is followed by two fully connected layers. Both networks generate an output of $T = 34$ poses where the first four poses are the initial poses. The loss is defined by:

$$L_G = \alpha \cdot L_G^{Huber} + \beta \cdot L_G^{NSGAN} + \gamma \cdot L_G^{style} + \lambda \cdot L_G^{KLD} \tag{5.37}$$

See [61] for an explanation of the individual loss terms.

Discriminator Network The Discriminator takes as input poses, either generated by the encoder or sampled from the recorded dataset. It outputs a scalar ($\mathbb{R}^1$) which encodes if the input is either artificially generated or comes from a real recording. The GRU version of the discriminator has three 1D convolutional layers followed by 4 GRU layers with varying sizes of their output dimensions. As the Generator, it has two fully connected layers on its top. The first has the same dimension as the GRU layer's output, and the second a dimension of 28. The Transformer network consists of 5 blocks of Transformer decoders with 3 heads and feedforward networks of dimension 128. As the GRU network, it is followed by two fully connected layers. Additionally to the poses, the Transformer network receives as input a positional encoding similar to the generator network.

The model is trained following the generator/discriminator update procedure, i.e. one update step for the discriminator followed by one update step for the generator. The discriminator loss is defined by:

$$L_D = -\mathbb{E}[\log(D(d))] - \mathbb{E}[\log(1 - D(\hat{d}))] \tag{5.38}$$

where d is the output of the discriminator and $\hat{d}$ is the ground truth if the input is either generated or real.

Experiments

We trained our proposed network model under various conditions (Table 5.1) on data of recorded gestures of human speakers from TED talks [62].

Experimental Procedure

The dataset is split in a training, validation, and test set with 200k, 26k, and 26k samples accordingly. Each sample consists of the spoken text during a 2 second audio recording and the corresponding upper body poses of a human speaker. Poses were recorded with a frame rate of 15Hz. Additionally, 4 poses before the start of the spoken text were recorded which are given to the networks so that their generated poses continue from these. As a result, each text sample has $T = 34$ corresponding poses.

Each training lasted for 300 epochs with a batch size of 512. An Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$ was used. The learning rate was set to $5.\text{e}{-4}$ for the generator and $1.\text{e}{-5}$ for the discriminator. We evaluated our models for different generator and discriminator dimensions, and different weights (α, β, γ, δ) of the loss terms (Table 5.1). As these results are preliminary evaluations, each model was trained for a single random seed.

[1] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

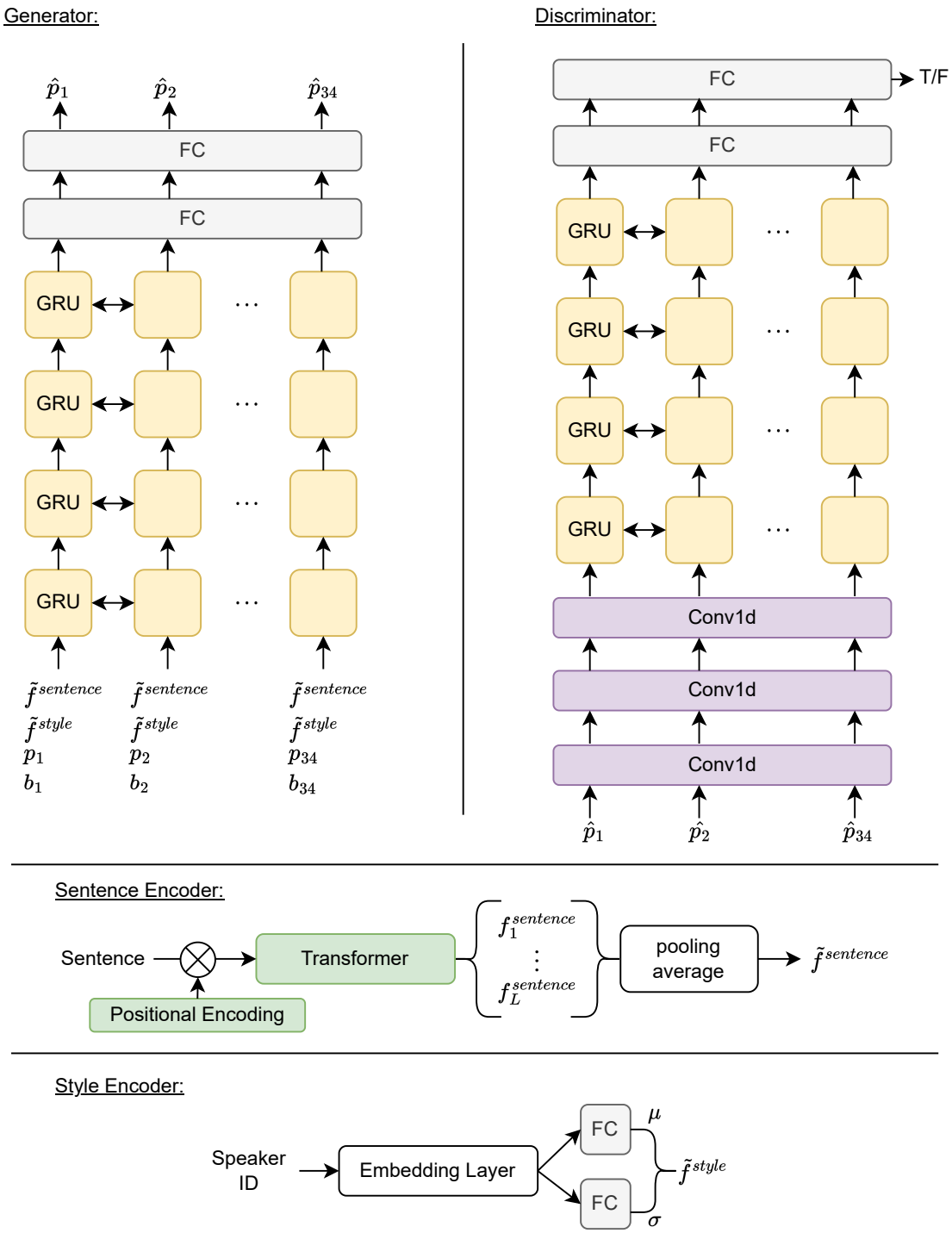


Figure 5.7: Architecture of our gesture generation architecture using GRU layers.

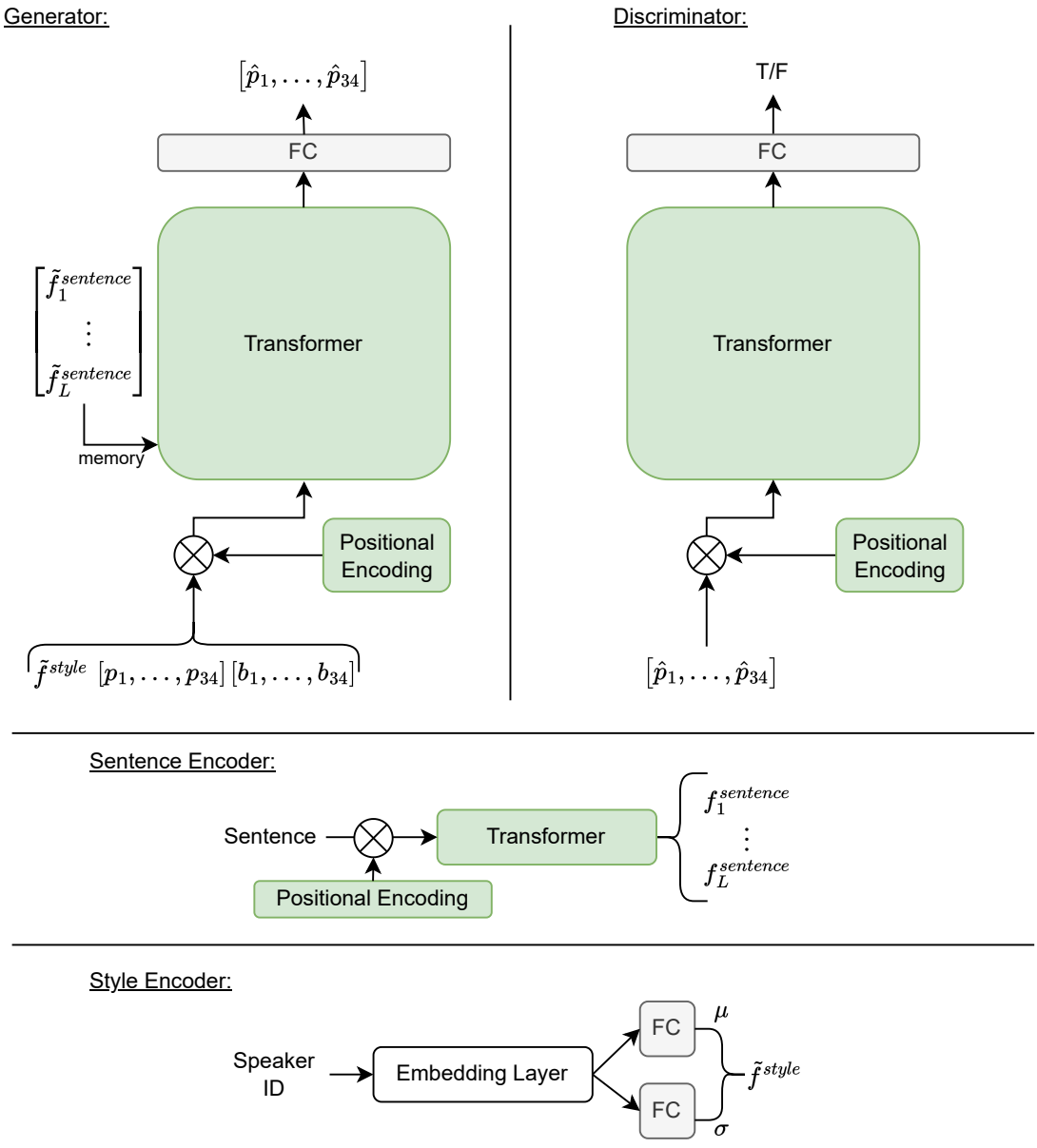


Figure 5.8: Architecture of our gesture generation architecture using Transformers.

Table 5.1: Settings and results for some of the preliminary evaluated gesture generation models.

Model	Gen Dim	Dis Dim	α	β	γ	λ	Epochs ↓	MAEJ ↓	FD ↓	FGD ↓
[61]	300	300	500	5	0.05	0.1	85	**0.0262**	**59.06**	3.45
gru 1	560	560	100	0.1	0.05	0.1	241	0.0308	66.24	**1.62**
gru 2	512	512	100	0.1	0.05	0.1	245	0.0300	65.43	1.63
gru 3	300	300	50	1	0.01	0	**45**	0.0313	66.09	2.58
gru 4	1024	1024	100	1	0.05	0.1	76	0.0315	66.68	2.03
trans 1	1024	128	100	0.2	0.05	2	126	0.0290	65.20	2.96
trans 2	1024	128	100	0.2	0.05	0.5	93	0.0280	64.54	2.97
trans 3	1024	128	100	0.2	0.05	0.2	90	0.0300	66.41	2.94

Results

Our main evaluation metric is the Fréchet Gesture Distance (FGD), as done in [61]. It is inspired of the Fréchet Inception Distance (FID) used in the image generation domain. It calculates the Fréchet distance between the distributions of real and generated gestures in a latent feature space. The latent feature space is computed by a autoencoder that was trained to recreate input poses. We use the pretrained autoencoder from [61]. The $FGD(X, \hat{X})$ is defined as the Fréchet distance between the Gaussian mean and covariance of the latent features of human gestures X and the Gaussian mean and covariance of the latent features of the generated gestures $\hat{X}$:

$$\mathrm{FGD}(X, \hat{X}) = ||\mu_r - \mu_g||^{1/2} + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\right) \tag{5.39}$$

where μ_r and Σ_r are the mean and variance of the latent feature distribution Z_r of real human gestures X , and μ_g and Σ_g are the mean and variance of the latent feature distribution Z_g of generated gestures $\hat{X}$.

A second evaluation metric is the Features Distance (FD) which is the mean average error between the latent features of the ground truth and the generated gestures:

$$\mathrm{FD}(X, \hat{X}) = ||\mu_r - \mu_g|| \tag{5.40}$$

The final evaluation metric is the mean average error of joint coordinates (MAEJ) between the ground truth poses $\hat{P}$ and the generated ones P:

$$\mathrm{MAEJ}(P, \hat{P}) = \frac{1}{D+T+N}\sum_{i=1}^{D}\sum_{j=1}^{T}\sum_{k=1}^{N} p_{ijk} - \hat{p}_{ijk} \tag{5.41}$$

where i iterates over data samples, j over poses in one sample, and k over the individual joints.

We report the FGD, FD, and MAEJ for the test set (Table 5.1). The MAEJ and FD are for the model parameters that resulted in the best FGD according to the validation set. Epochs denote the number of epochs before the best FGD per model was found. We compared our models to the model by [61]. Please note that this model also takes as input the audio recording which ours do not. All our proposed models obtain a better FGD, ranging from 1.62 to 2.97 compared to 3.45. For the MAEJ the original model yields a slightly better results of 0.026 compared to our models, ranging from 0.028 to 0.032. Also for the FD the original model achieved a better result of 59.1, compared to our models, ranging from 64.5 to 66.9. These results show that our models, are worse in regenerating the exact movements for a given ground truth (MAEJ, FD), but they are better in capturing natural motions which is better measured by the FGD ([61]).

Comparing the GRU (gru 1,2,3,4) with the Transformer (trans 1, 2, 3) models shows that the Transformer layers do not result in an improvement for this task. Their FGD is larger with 2.94 (trans 3) compared to 1.62 (gru 1), while the MAEJ (0.03 compared to 0.031) and FD (66.4 compared to 66.2) are similar. We could only observe that the learning process was more stable with the Transformer networks compared to the GRU networks, which often had dips in their loss over the course of learning.

Conclusion

We introduced a deep neural model for the generation of gestures. Our model takes as input the spoken text and ouputs a sequence of joint positions. Our model outperforms its succesor from [61] in terms of FGD by introducing a Transformer model for the encoding of the text sentence. Replacing its GRU units with Transformer layers did not result in a improved performance. The current results are still preliminary and we will evaluate further possible architecture variations and hyperparameters to improve the generation of gestures. Furthermore, we will evaluate the performence of the generated gestures on the physical robot.

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

6 Conclusion

In this deliverable we listed and described various deep architectures and their training procedures that were developed and integrated for the SPRING project. Their tasks range from the visual perception of humans, analysis and understanding of their utterances, generation of dialog, to the behavioral control of ARI. Besides their task variety, the developed networks also have various architectures, ranging from simple fully connected feedforward architectures, convolutional architectures, recurrent architectures such as GRU, to Transformers. The SPRING project proves to be a successful seedbed for new ideas and architectures in deep learning.

Please note, the current document provides only a snapshot of our current development of deep architectures. We will further investigate and improve upon the here discussed architectures throughout the SPRING projects life time.

Bibliography

[1] Neziha Akalin and Amy Loutfi. Reinforcement learning approaches in social robotics. *Sensors*, 21(4):1292, 2021.

[2] Kavosh Asadi, Ronald E Parr, George D Konidaris, and Michael L Littman. Deep rbf value functions for continuous control. *arXiv preprint arXiv:2002.01883*, 2020.

[3] Anand Ballou, Chris Reinke, and Xavier Alameda-Pineda. Variational meta reinforcement learning for social robotics. *arXiv preprint arXiv:2206.03211*, 2022.

[4] 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.

[5] David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[6] 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.

[7] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.

[8] Shlomo E. Chazan, Jacob Goldberger, and Sharon Gannot. Speech enhancement with mixture of deep experts with clean clustering pre-training. In *IEEE International Conference on Audio and Acoustic Signal Processing (ICASSP)*, Toronto, Ontario, Canada, June 2021.

[9] Weikai Chen, Xiaoguang Han, Guanbin Li, Chao Chen, Jun Xing, Yajie Zhao, and Hao Li. Deep rbfnet: Point cloud feature learning using radial basis functions. *arXiv preprint arXiv:1812.04302*, 2018.

[10] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE, 2017.

[11] Yuren Cong, Michael Ying Yang, and Bodo Rosenhahn. Reltr: Relation transformer for scene graph generation. *ArXiv*, abs/2201.11460, 2022.

[12] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural computation*, 5(4):613–624, 1993.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[14] 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.

[15] Clement A Gehring. Approximate linear successor representation. *Reinforcement Learning Decision Making*, 2015.

[16] Jia-Chen Gu, Chao-Hong Tan, Chongyang Tao, Zhen-Hua Ling, Huang Hu, Xiubo Geng, and Daxin Jiang. Hetermpc: A heterogeneous graph neural network for response generation in multi-party conversations. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 5086–5097. Association for Computational Linguistics, 2022.

[17] Jia-Chen Gu, Chongyang Tao, and Zhen-Hua Ling. Who says what to whom: A survey of multi-party conversations. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5486–5493. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Survey Track.

[18] Jia-Chen Gu, Chongyang Tao, Zhen-Hua Ling, Can Xu, Xiubo Geng, and Daxin Jiang. MPC-BERT: A pre-trained language model for multi-party conversation understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3682–3692, Online, August 2021. Association for Computational Linguistics.

[19] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[23] Nithin Rao Koluguri, Taejin Park, and Boris Ginsburg. Titanet: Neural model for speaker representation with 1D depth-wise separable convolutions and global context, 2021.

[24] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *International Journal of Computer Vision*, 123:32–73, 2017.

[25] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

[26] Stéphane Lathuilière, Benoît Massé, Pablo Mesejo, and Radu Horaud. Neural network based reinforcement learning for audio–visual gaze control in human–robot interaction. *Pattern Recognition Letters*, 118:61–71, 2019.

[27] 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.

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

[29] Xin Lin, Changxing Ding, Jinquan Zeng, and Dacheng Tao. GPS-Net: Graph Property Sensing Network for Scene Graph Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3743–3752, 2020.

[30] Hengyue Liu, Ning Yan, Masood S. Mortazavi, and Bir Bhanu. Fully Convolutional Scene Graph Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11546–11556, 2021.

[31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[33] Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[34] 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.

[35] Claudio Pedica and Hannes Vilhjálmsson. Social perception and steering for online avatars. In *International Workshop on Intelligent Virtual Agents*, pages 104–116. Springer, 2008.

[36] Sebastian Pineda-Arango, David Obando-Paniagua, Alperen Dedeoglu, Philip Kurzendörfer, Friedemann Schestag, and Randolf Scholz. Improving sample eficiency with normalized rbf kernels. *arXiv preprint arXiv:2007.15397*, 2020.

[37] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[38] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[39] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.

[40] Chris Reinke and Xavier Alameda-Pineda. Xi-learning: Successor feature transfer learning for general reward functions. *arXiv preprint arXiv:2110.15701*, 2021.

[41] Satoru Satake, Takayuki Kanda, Dylan F Glas, Michita Imai, Hiroshi Ishiguro, and Norihiro Hagita. A robot that approaches pedestrians. *IEEE Transactions on Robotics*, 29(2):508–524, 2012.

[42] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[43] SPRING Project. D2.1: Visual-based localisation in realistic environments. Link.

[44] SPRING Project. D2.2: Semantics-based localisation in realistic environments. Link.

[45] SPRING Project. D3.1: Audio-visual speaker tracking in realistic environments. Link.

[46] SPRING Project. D3.2: Audio-visual speaker tracking in relevant environments.

[47] SPRING Project. D4.1: Multi-modal behaviour recognition in realistic environments. Link.

[48] SPRING Project. D4.2: Multi-modal behaviour recognition in relevant environments.

[49] SPRING Project. D5.1: Initial high-level task planner and conversational system prototype for realistic environments. Link.

[50] SPRING Project. D5.2: Multi-party asr and conversational system in realistic environments.

[51] 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.

[52] 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, 2020.

[53] 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, 2020.

[54] Mohammed Suhail, Abhay Mittal, Behjat Siddiquie, Chris Broaddus, Jayan Eledath, Gerard Medioni, and Leonid Sigal. Energy-Based Learning for Scene Graph Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13936–13945, 2021.

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

[56] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

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

[58] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788, 2020.

[59] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[60] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. Fastdepth: Fast monocular depth estimation on embedded systems. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6101–6108. IEEE, 2019.

[61] Youngwoo Yoon, Bok Cha, Joo-Haeng Lee, Minsu Jang, Jaeyeon Lee, Jaehong Kim, and Geehyuk Lee. Speech gesture generation from the trimodal context of text, audio, and speaker identity. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[62] Youngwoo Yoon, Woo-Ri Ko, Minsu Jang, Jaeyeon Lee, Jaehong Kim, and Geehyuk Lee. Robots learn social skills: End-to-end learning of co-speech gesture generation for humanoid robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4303–4309. IEEE, 2019.

[63] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2018.

[64] Pourya Habib Zadeh, Reshad Hosseini, and Suvrit Sra. Deep-rbf networks revisited: Robust classification with rejection. *arXiv preprint arXiv:1812.03190*, 2018.

[65] Alireza Zareian, Zhecan Wang, Haoxuan You, and Shih Fu Chang. Learning Visual Commonsense for Robust Scene Graph Generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 642–657, 2020.

[66] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural Motifs: Scene Graph Parsing with Global Context. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5831–5840, 2018.

[67] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. Fairmot: On the fairness of detection and re-identification in multiple object tracking. *International Journal of Computer Vision*, 129(11):3069–3087, 2021.

[68] 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.

[69] Ming Zhong, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. Dialoglm: Pre-trained model for long dialogue understanding and summarization. *CoRR*, abs/2109.02492, 2021.

[70] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[71] Zhiqian Zhou, Pengming Zhu, Zhiwen Zeng, Junhao Xiao, Huimin Lu, and Zongtan Zhou. Robot navigation in a crowd by integrating deep reinforcement learning and online planning. *Applied Intelligence*, pages 1–17, 2022.

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