Perceiving, Planning, Acting, and Self-Explaining:

A Cognitive Quartet with Four Neural Networks

by

Yantian Zha

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved June 2022 by the
Graduate Supervisory Committee:

Subbarao Kambhamapti, Chair
Baoxin Li
Siddharth Srivastava
Jianjun Wang

ARIZONA STATE UNIVERSITY

August 2022

ABSTRACT

Learning to accomplish complex tasks may require a tight coupling among different levels of cognitive functions or components, like perception, acting, planning, and self-explaining. One may need a coupling between perception and acting components to make decisions automatically especially in emergent situations. One may need collaboration between perception and planning components to go with optimal plans in the long run while also drives task-oriented perception. One may also need self-explaining components to monitor and improve the overall learning. In my research, I explore how different cognitive functions or components at different levels, modeled by Deep Neural Networks, can learn and adapt simultaneously. The first question that I address is: Can an intelligent agent leverage recognized plans or human demonstrations to improve its perception that may allow better acting? To answer this question, I explore novel ways to learn to couple perception-acting or perception-planning. As a cornerstone, I will explore how to learn shallow domain models for planning. Apart from these, more advanced cognitive learning agents may also be reflective of what they have experienced so far, either from themselves or from observing others. Likewise, humans may also frequently monitor their learning and draw lessons from failures and others' successes. To this end, I explore the possibility of motivating cognitive agents to learn how to self-explain experiences, accomplishments, and failures, to gain useful insights. By internally making sense of the past experiences, an agent could have its learning of other cognitive functions guided and improved.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor Prof. Subbarao Kambhampati for his support and advice on my research, career, and life. I have had the dream of exploring valuable directions to combine Artificial Intelligence and Robotics. Because of the freedom, support, and encouragement from my advisor, I have enjoyed researching a variety of problems in the intersections of planning, perception, learning, natural language processing, and robotics, throughout my PhD. Prof. Kambhampati has supported me much beyond as a research advisor – he does not only care about my research but also how my career will evolve, what limitations do I still have, and my overall well-being. I feel very fortunate to be one of the few students in the world who have Prof. Kambhampati as their PhD advisor.

Thank you to my committee members Prof. Baoxin Li, Prof. Siddharth Srivastava, and Dr. Jianjun Wang. I have collaborated with Prof. Li on a planning-and-vision project for a long time. I greatly appreciate Prof. Li for his support and guidance which help me accomplish multiple publications in the intersection between planning and vision. The experience builds a strong foundation for my robotics research later. Taking Prof. Srivastava's robotics class is one of my most memorable course experiences. I sincerely thank Prof. Srivastava for many fruitful conversations throughout my PhD. I heartily appreciate Dr. Wang for mentoring me at ABB when I did an internship there. Dr. Wang tremendously improved my research ability in robotics during the five and a half months internship! I appreciate my undergraduate advisors Prof. Xudong Ma and Prof. Kun Qian for getting me into the world of robotics which was the starting point of my trip to pursue my dream in robotics.

I would like to thank Prof. Yu "Tony" Zhang for mentoring me on how to find interesting research problems and develop robotics software when I was a MS student at Yochan lab. I am grateful to Prof. Hankz Hankui Zhuo for collaborating with

me on a series of works that use neural networks and natural language processing techniques to solve planning problems. The experience helps me understand deeper the deep neural networks and how they could be connected to planning.

I am thankful to many other professors at ASU: Prof. Yezhou Yang and Prof. Heni Ben Amor who have had helpful discussions of my research projects. I am also very honored to be invited by Prof. Yezhou Yang to give multiple talks in his group.

I would like to thank my current and graduated colleagues at Yochan Lab: Minh Do, Tathagata Chakraborty, Lydia Manikonda, Sailik Sengupta, Anagha Kulkarni, Sarath Sreedharan, Sachin Grover, Sriram Gopalakrishnan, Alberto Olmo, Zahra Zahedi, Utkarsh Soni, Lin Guan, Mudit Verma, Siddhant Bhambri, Karthik Valmeekam, Denis Liu, Niharika Jain, Aditya Prasad Mishra, Daniel D'Souza, and Gabriel Saba. Among them, I owe the most to Sarath Sreedharan who is also my roommate. Thanks to Anagha Kulkarni who invited me into her project, and then I have my first robotics paper. The first Yochan member that I met was Daniel D'Souza who was playing with our PeopleBot. Thanks to Daniel, my robotics debugging skills are further improved. Sriram Gopalakrishnan is a great writer and presenter. My collaboration with Sriram Gopalakrishnan greatly improved my writing and presentation skills. Tathagata Chakraborty and Lydia Manikonda were the most senior lab members when I joined. Their huge amount of citations motivates me to keep being productive. I appreciate Lin Guan for contributing to many of my papers. Our collaboration has inspired me to improve my research philosophy. I appreciate Siddhant Bhambri for offering critical comments that inspired me to improve our algorithm. I am also thankful to my collaborators Mudit Verma and Denis Liu. The project that we are involved is really helpful for me to think about how to become a better mentor and advisor. Besides those people, I am also thankful to some students in the Visual Representation and Processing Group: Yikang Li and Tianshu Yu who collaborated with me on

planning-and-vision projects.

I am grateful to many support staff at Arizona State University: Arzuhan Kavak, Jaya Krishnamurthy, Monica Dugan, Pamela Dunn, Christina Sebring, Brint MacMillan, Nicholas Beck, Theresa Chai. I am also thankful to my past and current colleagues at ASU: Yuhan Sun, Ruocheng Guo, Lu Cheng, Jundong Li, Liang Wu, Kewei Cheng, Sicong Liu, Kai Shu, Kaize Ding, Xue Hu, Imane Lamrani, Zhen Zeng, Ze Gong, Shuai Li, Xin Ye, Zhiyuan Fang, Man Luo, Pulkit Verma, Naman Shah, Rashmeet K Nayyar, Rushang Karia; My friends who helped me when I was an MS student at ASU: Xiaoqin Xue, Liangyu He, Mengyuan Zhang, Haoran Lv, Taeyeong Choi; My colleagues at ABB: Biao zhang, Yinwei Zhang. In particular, thanks Yuhan Sun and Ruocheng Guo for your encouragement and counseling whenever I needed.

Finally, this thesis is dedicated to my parents, Tao Zha and Guanqun Luo, for your love at support in my life.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

If you have a car, remember how you learn to drive it. You probably start by learning how to observe your local environment effectively (attention) and efficiently (attention control). You world learn to coordinate different observing actions like checking your rearview or side mirrors or attending the front or areas on both sides. Next, you would learn how to react in different situations. Are you in the blind spot of another vehicle? What should you do when you want to change lanes when there is an approaching vehicle behind? How to handle different road signs? Finally, you probably would learn to plan and execute the best route. This could be learned from your experiences if you have stayed in an area long enough (e.g. you know when are peak hours for a road), or from learning to use navigation tools like Google Maps together with visually searching for landmarks that match your route plan.

During the process of learning these, you sometimes would introspect to improve yourself. Consider the two examples in Figure. 1.1. Assuming you already learned how to turn right when there is no obstacle, by observing how close you and the road corner are and making a circular turn at a proper time. However, when you meet a new situation as in the Figure. 1.1.a (there is a truck and the road is narrow), you make a right turn just as how you used to do and hit the edge of the truck. You may then think about why you fail this time, what are the differences between how you make the right turn this time that you fail and the last time when you were successful. Maybe you should observe the truck edge instead of the road corner, and only use the truck edge as the reference to your decision-making? Based on this kind of retrospective thinking, you may make a plan to practice to handle such a situation.

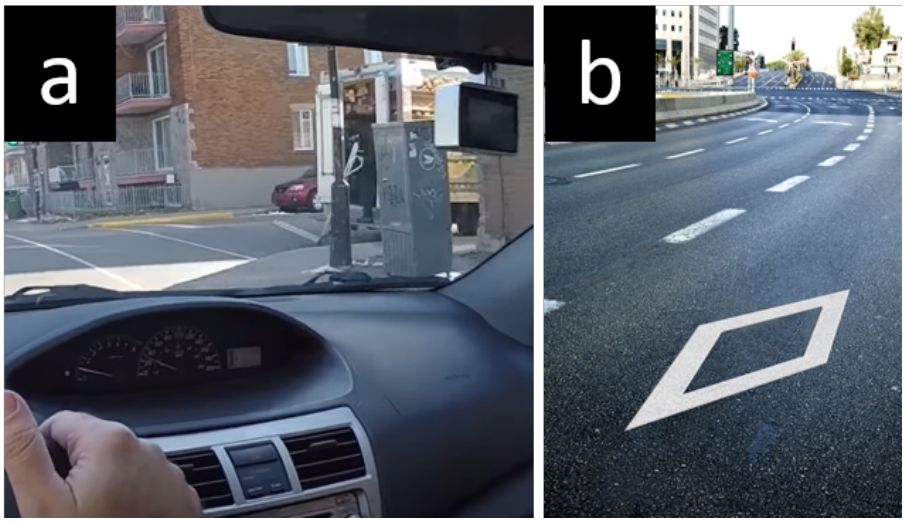


Figure 1.1: An Illustrative Example of Learning to Drive.

Eventually, you would handle a new subskill of turning right: make an L-turn.

Another example is in the Figure. 1.1.b. Imagining when you observe your friend's driving, and she/he always try to avoid driving into a lane marked with a diamond symbol that you don't know the meaning of. Your friend's behavior is mismatching to your current knowledge. By self-explaining this, you may conclude that the diamond symbol has some kinds of negative effects on driving. After having this knowledge, when you drive next time and see a lane with such diamond symbols again, you would also avoid driving into that lane.

The above example illustrates how you, as a human, may use your cognitive intelligence during the process of learning something difficult. Human intelligence, most of the time, is the biggest resource to inspire the development of Artificial Intelligence (AI) theories. The ultimate goal of researching AI is to make machines as smart as humans, according to Alan Turing and his famous Turing Test. To be more specific, such smart machines should have these capabilities: natural language processing, knowledge representation, automated reasoning, machine learning, computer vision,

and robotics ([126]).

In the old days before the rise of Deep Neural Networks, AI researchers kind of isolatedly contributed to those subfields. For example, Standford Research Institute Problem Solver (STRIPS) is a classic planner whose theory is closely connected to automated reasoning and knowledge representation. Applications for problems in the fields of natural language processing and computer vision usually take advantage of machine learning theories like Hidden Markov Models (HMMs) or Support Vector Machine (SVM).

However, the learning-to-drive example suggests that those AI fields could have more and tighter interconnections, especially with the rise of Deep Neural Networks that mimic how humans' neurocognitive functions work. While deep learning is essentially a machine learning technique in large part inspired by neuroscience, the fact that it can be effectively trained as a super-powerful nonlinear function approximator by large-scale datasets leads to more cross-fertilization among those sub-fields in AI. Heretofore, deep learning is the dominating method in Computer Vision and Natural Language Processing communities. Beyond this, most recently people have also applied deep learning to solve problems that span more broadly across multiple sub-fields (e.g. Visual Question Answering).

When we consider Deep learning from an AI perspective, we would like to ask what it can bring to enhance AI agents. Deep Learning can enhance a traditional AI learning agent by improving the performances of each component separately (e.g. sensing, reasoning, or planning). However, Deep neural networks that are inspired by how animal brains work naturally have tight connections to cognitive psychology and neuroscience. From a cognitive psychology perspective, cognitive functions like perception, attention, reasoning, planning, and acting are interleaved that contribute together to agents' intelligence, according to [140].

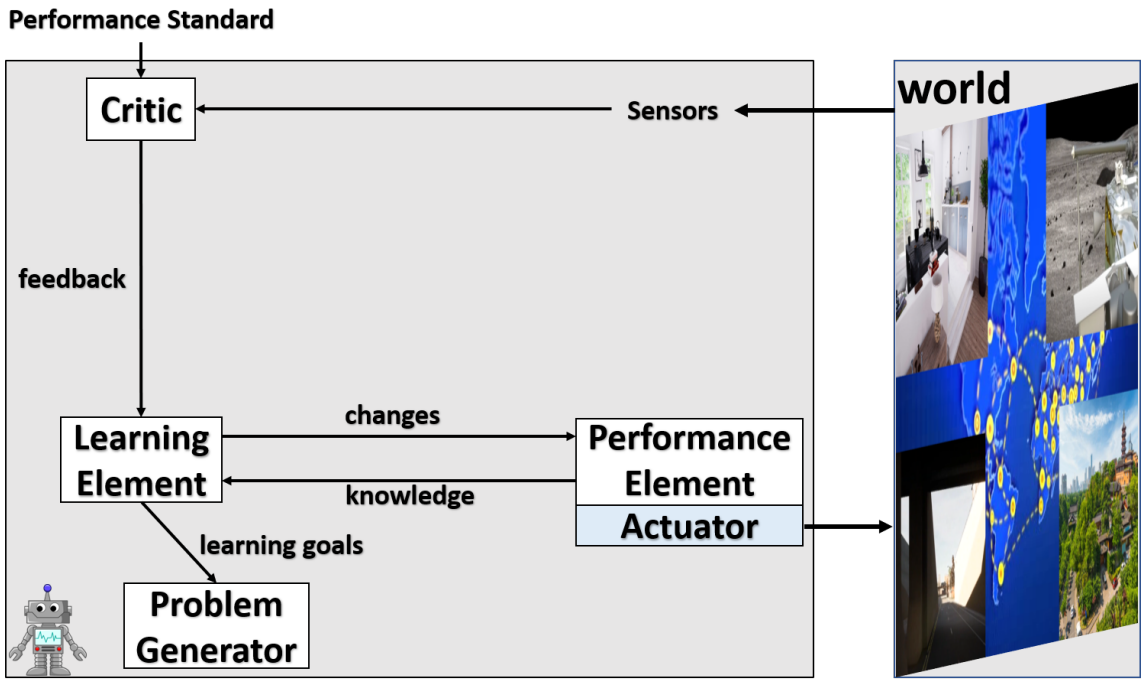


Figure 1.2: A Traditional Learning Agent (from [126])

1.1 A Traditional Learning Agent

The idea of obtaining intelligence by learning instead of programming for AI agents was proposed by Turing. A traditional learning agent is illustrated in Figure. 1.2 ([126]). The framework describes how an agent could improve its functions by learning from its experiences of behaving in an environment. The learning agent framework has four basic conceptual components: Learning element, Critic, Performance element, and Problem Generator.

1.1.1 Critic

The role of the Critic is to evaluate the current performance of the learning agent. The evaluation requires applying some kind of predefined performance criteria to the

current agent states and the observation of the world. The performance criteria can be seen as a human-given knowledge, like loss functions, rewards, etc. The evaluation from the Critic is then used by the learning agent as a training signal.

1.1.2 Learning Element

The Learning element takes feedback signals from the Critic component to improve all functional elements like the Performance element. For example, suppose that the Critic is a reward function, the Learning element could use the rewards to learn a value estimator that predicates the value of taking an action in a state with long-term consideration. Then with this value estimator, the Performance element could select the best actions in the long run. By using feedbacks, the Learning element enables the agent to be adaptive and robust to environment changes. In this thesis, almost all learnable elements are formulated by parameterized models (e.g. deep neural networks).

1.1.3 Performance Element

The Performance element is essentially responsible for acting in the world based on current environment observation, which influences the world and what would be sensed the next. For example, a navigation robot has its motor controllers as Performance element. Its motor controllers may take the information processed from camera images and output linear and angular velocities, which in turn change the state of the robot and give the robot new sensations.

1.1.4 Problem Generator

With the Learning element, the Performance element would be able to behave optimally based on the agent's model. This kind of optimality matches the past

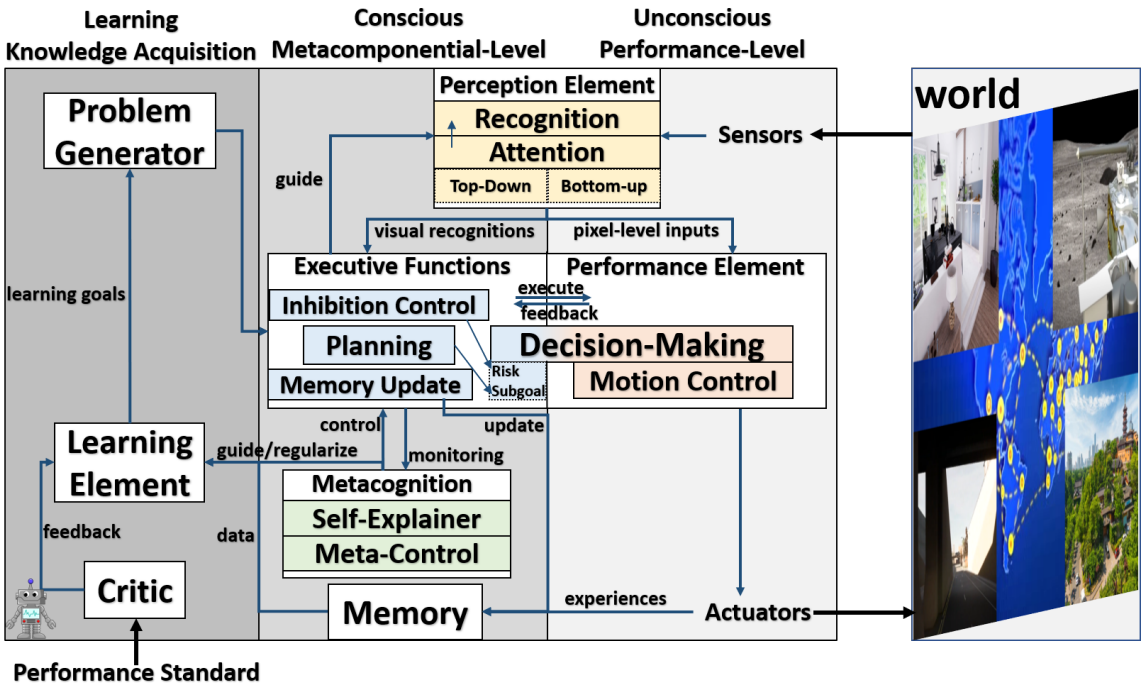


Figure 1.3: A Cognitive Learning Agent With Performance-level Cognitive Functions Like Perception and Actuation Elements, and Metacomponential-level Cognitive Functions Like Metacognition and Executive Functions.

experiences that the agent has collected. However, due to incomplete explorations and/or potential environmental changes, the agent's model may not be fully correct. This means there could have a better action that goes beyond the current knowledge of the agent. In this sense, the Problem Generator is responsible for the agent's exploration and obtain new experiences with unseen states, or actions that the agent does not think to be optimal.

1.2 A Revised Cognitive Learning Agent

According to [140], cognition is the ability to process information and react accordingly. To process information, humans or animals need to perceive, learn, remember,

and plan/act accordingly. More advanced cognitive agents may also need to consciously think about what they need to perceive (i.e. top-down attention), or why what they did led to good/bad outcomes, etc. [140] concludes from The Triarchic Theory of Intelligence ([141]) that cognition lies the center of intelligence, and the information processing in cognition is a highly interdependent corporation among three functional components: metacomponent (metacomponential component), performance component, and knowledge-acquisition component. The metacomponential component is responsible for higher-order processes like planning, monitoring, and evaluating in the problem solving processes of agents. The performance component is involved in processes like sensing and acting that are directly related to performance. The knowledge-acquisition component aims at learning how to solve problems so that the agent can keep improving itself and adapt to new changes. The metacomponential component includes higher-order cognitive functions, and therefore is more conscious, slower in processing, and consumes more resources like memory ([80]). In contrast, the cognitive functions included in the Performance component are more tacit and unconscious. As reviewed in [80; 17; 122; 121], more recent cognitive psychological studies (especially after the year 2011) further divide metacomponential component into two highly-interdependent parts: Metacognition and Executive Functions. Metacognition can be understood as "thinking about thinking", which deals with meta-level tasks like managing or strategizing someone's thinking (introspecting), monitoring the learning or problem-solving, retrospecting, and self-evaluating. Executive Functions are high-order cognitive functions that are goal-directed, self-regulative, and adaptive ([121]). To achieve Metacognition, you would need to call some of your Executive Functions like planning and reasoning.

Current researches that connect cognitive learning and neural networks mostly dive into knowledge-acquisition components and performance components. They

more or less overlook metacomponential components (Executive Functions and particularly Metacognitive functions). While there are different subclasses of Executive Functions and Metacognition functions, in this thesis, I would focus on planning and plan recognition as Executive Functions, and a novel Metacognition process, called self-explanation. Self-explanation is an important Metacognition function, which is also connected to the current popular literature of interpretable AI.

1.2.1 Perception Element

The Perception element in Figure. 1.3 is different from the Sensors module in the traditional learning agent framework (Figure. 1.2). The Perception element may further have an attention mechanism. Bottom-up attention is an automatic process that an agent focuses on a seemingly interesting region without any deliberate thinking. Top-down attention typically means that the attention is intentionally guided or biased by cognitive processes that require extra mental efforts, like memory-calling, reasoning, etc., as described in [140].

Classic visual sensations are automatic and unconscious. The Performance element takes processed visual signals from the Perception element to make decisions. The Perception element may also provide visual recognition outputs for functions that require a state abstraction of the world (e.g. motion control for grasping). However, when perception is conditioned on conscious cognition functions, it becomes a conscious process too. A planning system could actively adjust perception configurations like resolutions, view of fields, view angles, etc. A plan recognition system may also consciously guide the top-down attention.

1.2.2 Executive Functions

Executive Functions are high-order cognitive functions that are conscious, more controlled, and sometimes self-regulated. Typical Execution Functions includes planning, decision-making, inhibiting, and memory updating. Inhibition is an ability to interrupt or inhibit an automated behavior for avoiding risks or gaining benefits ([122; 121; 80]). In this thesis, I mainly focus on planning, plan recognition and conscious decision-making.

Planning

When an agent does planning, it typically tries to find a sequence of high-level actions given a domain model, an abstracted description of initial states, and goal constraints. Such an action sequence would lead the agent to go from the initial state to the goal. Therefore, this action sequence would be the solution of a planner, which takes as input a task description (initial and goal states) and action models (models for describing preconditions and effects of available actions). The simplest version of planning is deterministic planning, which means after applying an action in the current state, the agent can deterministically reach a new state. Note that parts of the new state is the effect of applying the action, which can be modelled as a set of predicates with satisfiable arguments in PDDL language ([3]) or terminal conditions in Options framework ([143]).

Planning is naturally connected to the Perception element. Most classic planning works assume that there is an optimal visual recognition module, which could tell if an effect is satisfied or not. When you drive a car, you have your navigation plan, which is a sequence of high-level actions and subgoals (effects) with some kinds of landmark information (e.g. go forward until you see X avenue). You would visually check if your

surroundings satisfy the effect or not of current action in your plan. If satisfied, which means you have reached the current subgoal, you would probably continue to execute the next action and keep visually checking the corresponding effect's satisfiability again. Thus, planning is also naturally connected to the Performance element, so far as you need to perform in the world.

Decision-Making (Conscious)

Different from planning that finds an activity sequence between initial state and goal state, the decision-making process is about taking an action given a state. The action does not necessary to be a high-level activity. It could also be as low-level as control commands. A conscious decision-making process would rely on not only states but also some high-level feature output from higher-order cognitive functions like planning or inhibition controlling. For example, the former could give subgoal features to guide the decision-making, and the latter could provide risk predictions to self-regularize the decision-making.

1.2.3 Metacognition

According to [95; 96], a fundamental metacognitive model can be formulated as a two-level structure: a meta-level model and an object-level model. The object-level model includes all basic cognitive functions. The meta-level model can monitor and metacognitively-control (meta-control) the object-level model. Monitoring and meta-control are two opposite directions. Monitoring allows object-level information goes up to the meta-level model, while meta-control allows the meta-level model to guide, influence, and control the functioning/learning of the object-level model. In this thesis, I use the term metacognition model to represent the meta-level model, and other conscious processes (e.g. executive functions) to represent object-level cognitive

functions. Unconscious cognitive processes are at the lowest-level (performance-level). As illustrated in Figure. 1.3, monitoring allows metacognition model to track and evaluate the learning and functioning of lower-level cognitive functions, like executive functions. For example, with monitoring and self-evaluation, the metacognition model could know what plan or action makes the agent perform well or into failures. Then more introspective processes could happen in the metacognition model to acquire meta-level knowledge like why it caused a failure. With such knowledge, the agent could meta-control those executive functions.

Inspired by the ongoing popularity of interpretable AI/ML as well as the self-monitoring, introspection, and retrospective self-evaluation processes in metacognition, in this thesis I introduce self-explanation, which can be seen as a novel metacognition mechanism for a neural network based learning agent to better improve itself (via meta-control). A similar idea of formulating metacognition as self-explanation for enhancing the learning of a traditional AI and learning agent is proposed in [27]. When an AI system meets a performance failure, the system is required to self-explain **how** and **why** the error happened. Then based on this explanation, the system would decide on what to learn (learning goals) and a better order of several learning algorithms to be tried (learning strategies). In contrast to this work, this thesis views the self-explanation from a more general perspective of connecting cognitive psychology and modern machine learning based on deep neural networks.

Self-Explaining

Self-explanation is an important metacognition that can guide the Learning element and improve the agent by itself. Self-explanation has relevance to the explanations introduced by the state-of-the-art interpretable AI/ML works. Both self-explanation and explanation aim at determining how and why some parts of inputs, some factors,

etc. lead to a model output (decision). The difference is that self-explanation should be used by the agent itself to improve its further learning, but an explanation from an interpretable AI model aims at making humans know more computational details of the AI model.

The intuition behind using self-explanation to support the learning of agents is simple: if you can explain it well, you should also learn it well. Embedding self-explanation in learning forces the learning agent to simultaneously learn to predicate an output, and learn to recognize "how" and "why" it makes that prediction. The latter could provide extra assistance/guidance to the learning of that agent, and make the agent more robust to environmental noise or domain change.

Meta-Control

The meta-control component should work with metacognition processes like self-explainer (which monitors lower-level processes) to generate meta-control signals to manage lower-level cognitive processes. For example, in the work [27], the generated explanation of a failure helps construct a learning strategy that could specify specific learning goals or constraints. In this thesis, I will show that an agent uses self-explanation to trigger meta-control signals like a predicted subskill that seemingly fit the current situation the best. This meta-control signal would then guide executive functions like conscious decision-making and regularize the Performance element. This finally shifts the distribution of experiences and guides the Learning element.

1.2.4 Learning Element

Similar to the traditional learning agent, the Learning element in this cognitive learning agent framework is responsible for taking the guidance from the Critic function and improve all trainable functions. To train those cognitive functions, the

Learning element also needs to retrieve data stored in the memory. The data typically are batches of stored experiences that the agent has collected via acting in the world.

In this thesis, I will show in Chapter 3 how to design a neural network to approximate planning functions, by learning shallow planning domain models. In Chapter 4, I will show how planning and the Perception element improves each other.

Here I would highlight the relationship between metacognition and the Learning element, which would be explained in detail in Chapter 5. Metacognition model could guide the learning of lower-level cognitive processes by adding extra feedback signals. Metacognition model could also directly send control signals to adjust those executive functions.

1.2.5 Performance Element

Performance element includes all automated (unconscious) processes required to construct a complete loop from environment to sensing, from sensing to acting, and from acting to the environment again.

Decision-Making (Unconscious)

An unconscious decision-making process directly maps sensing information or unconscious perception to low-level control commands, without searching a model, memory, or calling any higher-order conscious processes. Unconscious decision-making happens much more frequently than conscious decision-making in everyday life. Again taking the learning to drive as an example, most of your decisions that you make when you are driving are unconscious. You would not have time to think about if braking is reasonable when you are about to hit someone.

Motion Controller

Motion Controller contains unlearnable performance processes for precise speed, position, and torque control. The Motion Controller sends low-level control signals to the drive and motors in Actuators.

For example, in an industry, a sensing process with a depth camera sends a target pose estimation to the motion controller of an arm. Then the motion controller calculates a trajectory via certain kinematics solvers (e.g. an inverse kinematics solver). A trajectory usually comes in the format of a sequence of velocities and accelerations for all revolute joints per time step. At each time step, the corresponding velocities and accelerations data are used. Each velocity and acceleration value matches a revolute joint. After applying those velocities and accelerations to all revolute joints the desired motion can be done.

Motion Controller can be connected to the decision-making process by using output actions from it, instead of calculating trajectories. Note that in this case we assume the output actions follow the format of low-level control commands.

There could also have a connection between Motion Controller and Executive Functions (e.g. a planner), as in the works [67; 171]. The planner could work together with the Perception element to properly configure the motion controller. For example, a high-level action from the planner could be "pick_up_mug", then the Perception element uses this information to recognize the pose of a mug. Then the pose configures the motion controller to calculates a trajectory and execute it. Finally, the trajectory execution feedback could be sent back to the planner to decide if it should go to the next high-level action or redo the current one. A video demonstration for this example can be accessed at `https://bit.ly/2JweeYk`.

1.3 Dissertation Overview

My research aims at helping the learning of cognitive functions like perception, planning, and acting to be simultaneous, more tightly connected, and improved as a whole based on deep neural networks. I also specifically investigate if self-explaining, which can be viewed as a meta-level cognitive function, could further improve the overall learning of an agent. Therefore, the primary contribution of the dissertation is to leverage deep learning to achieve the following four goals:

- Integrating Perception and Acting (Chapter 2)
- Learning Shallow-Models for Planning Domains (Chapter 3)
- Integrating Perception and Planning (Chapter 4)
- Self-Explaining to Support the Overall Learning of an Agent (Chapter 5)

To enable a natural combination between the learning of a planning component and that of other cognitive functions, cornerstone research is how to reformulate symbolic classical planning problems as neural network based learning problems. This will be explained in Chapter 3. This further allows a more general and realistic view from a visual-plan-recognition perspective: grounded activities for planning might come from noisy and erroneous visual recognitions. In such cases, a more robust learning algorithms for learning shallow planning domains would be required.

In Chapter 4, I consider how a perception module and a plan recognition module could be tightly coupled. I investigate if a plan recognition model can be used to improve the learning of a visual recognition task.

In Chapter 5, I propose and explore how to embed self-explaining functions into learning to support the overall neural networks training.

Finally, in Chapter 6, I will conclude this dissertation with a summary of my contributions, some interesting alternative perspectives of viewing my research, and future directions based on self-explanation guided robot learning (Chapter 5) and advisable cognitive agents learning.

Chapter 2

COUPLING BETWEEN PERCEPTION AND ACTING

2.1 Cognitive Perception for Acting

Directly taking an action (or making a decision) based on the current visual observation of the world is important for accomplishing many valuable tasks in real world environments. Typical tasks include robotic grasping, autonomous driving, and playing video games. In such tasks, it would be difficult to specify or capture a clean state representation of objects that benefit task accomplishments. Imagine when a robot needs to grasp objects in different shapes and colors. Colors may hint at different textures and shapes may hint at geometric characteristics. Both are important for grasping but are hard to be described in a clean state representation like object poses.

Many researches have been conducted in the fields of Reinforcement Learning, Imitation Learning, and Robotics. All of such works in recent years tend to propose various convolutional neural networks (ConvNets) based policies to handle pixel abstraction. The mapping from pixels to actions are learned in various learning frameworks, which can generally be categorized into Imitation Learning and Reinforcement Learning. Imitation Learning is a supervised learning framework that learns a policy model by minimizing the distance between its predication and ground-truth action per step. On the other hand, Reinforcement Learning seeks for taking actions that could potentially maximize a reward objective based on neural network approximation of Q values.

In this chapter, I would introduce my work of contrastively learning visual attention as affordance cues from demonstrations for robotic grasping ([164]), which can

be seen as a novel way of acting in the world based on task-driven perception. The discovery of such affordance cues requires agents to deliberately discover distinctiveness underlying different expert demonstrations that accomplish a task in different ways (and therefore under different affordance assumptions).

2.1.1 Chapter Highlights

- In Sec. 2.2, I will show that imitating both the sequential decision-making and visual affordance knowledge from human outperforms merely imitating the former in robotic grasping tasks. The perception of visual affordance is driven by decision-making which makes the perception cognitive.

2.2 Learning Visual Attention as Affordance Cues from Demonstrations

Humans tend to understand objects and their parts from potentially applicable actions or motion primitives that can achieve effects for accomplishing a task. This phenomenon is abstracted as an ecological psychology concept called affordance established by J. J. Gibson ([44]). An affordance defines a mapping from an object feature to all applicable actions ([111]). Essentially, an affordance represents an object-action-effect relationship, which is an interactive procedure between an actor (e.g. a hand) and an object (e.g. a mug). Consider the example shown in Fig. 2.1, in a mug grasping task, a human teacher's affordance biases (affordance-effect judgments) might vary with the shape and size of the mug – the mug-A is graspable from its handle, whereas the mug-B is graspable from its body. When a robot learns from human demonstrations, it would be beneficial if the robot also discovers such affordance bias behind human demonstrations and generate (visual) affordance cues to support its learning.

There are several works ([29; 144; 31]) that propose to learn affordance knowl-

edge from human demonstrations that avoid a labor-intensive process of collecting affordance labels. However, previous works fully decouple affordance discovery from behavior learning and execution, which means the affordance predictor and the robot controller are trained or constructed separately. In those methods, the predicted affordances are fed into classic motion planners in an engineered fashion. In this work, we instead combine the learning of affordance knowledge and motion generation from human demonstrations in an end-to-end deep imitation learning framework. We further argue that learning visual attention as affordance cues, rather than explicitly modeling affordances, is enough to be a reliable latent feature for the actor. The actor here is also a part of the whole neural network, instead of being a separate and unlearnable external planner.

When learning affordance knowledge from demonstrations, visual attention can be an informative cue for inferring affordance. In fact, the close association between visual attention and affordance has been investigated by some cognitive psychology works ([8; 66]). According to them, humans use visual attention conditioned on objects' geometric and spatial properties to speed up affordance-effect judgments, which then helps generate motor signals (behaviors). When the attention comes from a controlled mental process (i.e. top-down attention) driven by a task, object parts that are permittable for accomplishing the task would be highlighted, from which proper affordance-effect judgments can be derived ([120]). Indeed, humans do not explicitly think about what are all possible ways of picking-up a mug at each of its parts or pixels once she or he already learned how to grasp different mugs. An earlier work ([85]) also investigates how visual attention could be associated with affordance cueing in the context of robotics.

In this work, we hypothesize that by encouraging the robot to attend to discriminative features that explain the differences between different demonstrated behaviors,

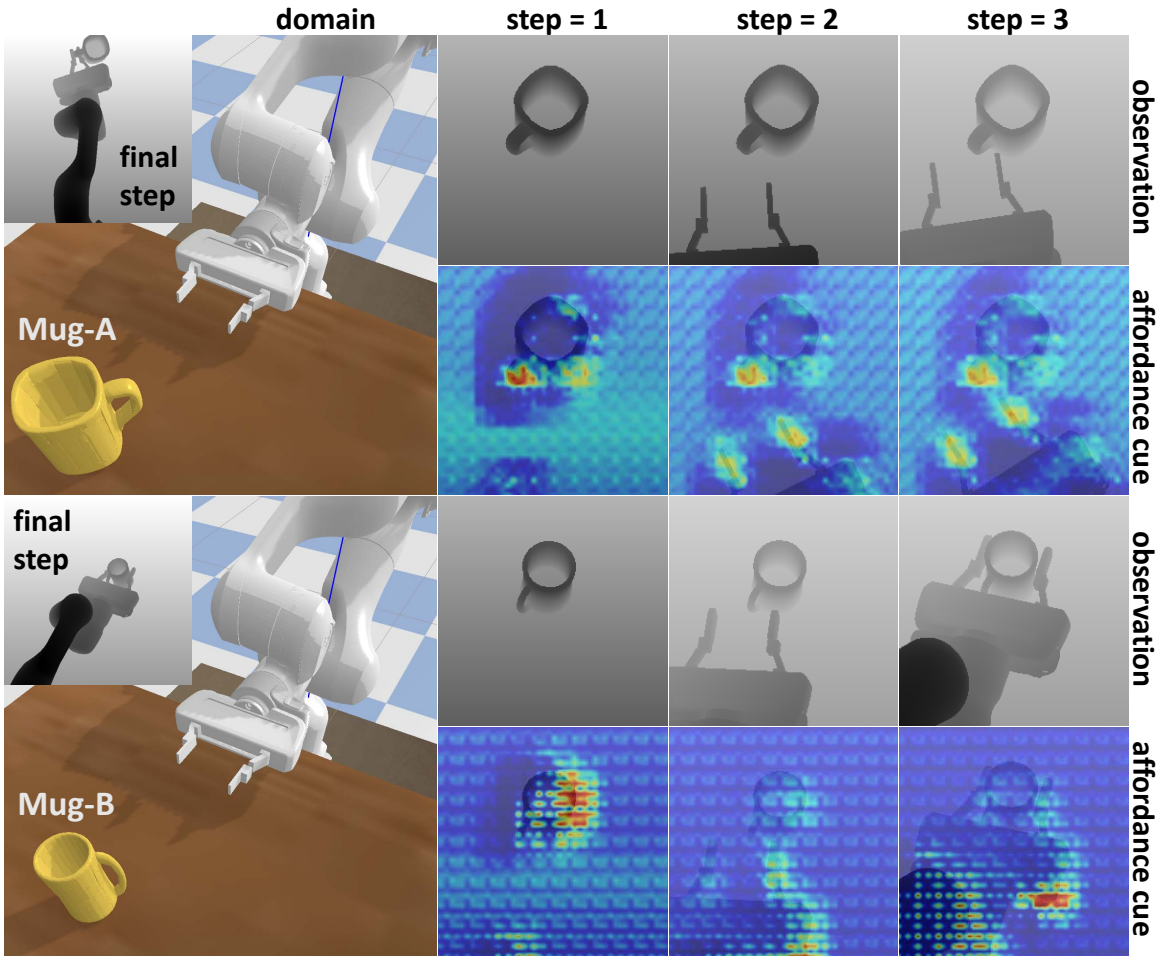


Figure 2.1: An Example of Two Mugs: Humans Should Have Different Affordance-effect Judgments on Their Bodies or Handles. At Each Step, the Robot Takes One Depth Image (Observation) and Generates One Attention Map, a.k.a Affordance Cue, That Further Triggers an Action. We Also Show how the Robot Eventually Picks Up a Mug, Either By Holding the Body or Handle, By Showing the Final Step Observation.

the robot will be able to more effectively discover affordance information and better imitate human behavior. This is because when a human teaches affordance knowledge to a robot, the human tends to think about what makes he/she have different affordance-effect judgments and how could the resulting trajectories be accordingly

different. Again taking Fig. 2.1 as an example, if the human shows two different trajectories for the robot to pick up the two different mugs, the robot could discover important discriminative features regarding mug shapes and sizes and consequently better learn from demonstrations. If the robot could attend to the handle of Mug-A, then it would be more likely for the robot to grasp the Mug-A by its handle, rather than its body.

Therefore, our central problem is to help robots to imitate humans not only at the behavior level but also with a hidden objective of discovering the affordance-relevant distinctiveness underlying human demonstrations. As such, robot could attend to appropriate parts of a specific mug that hints on the same affordance in human's mind and therefore helps trigger a similar behavior to humans'. To address the problem, we propose to use a deep Siamese encoder and trajectory decoder that are trained jointly with a contrastive loss and a behavior cloning loss in an end-to-end fashion. We also propose a coupled triplet loss to encourage the discovered discriminative features to be more affordance-relevant.

To thoroughly evaluate our work, we compare our model with a variant version that uses a normal triplet loss, a version without using Siamese network, and a baseline ConvNet-based behavior cloning model. We evaluate those models in terms of the grasping success rates and visualizations of predicted affordance cues. We empirically show that our model with the coupled triplet loss performs the best.

To the best of our knowledge, our work is the first to combine grasping affordance learning and imitation learning from expert demonstrations based on deep neural networks. The video that demonstrates the advantage of our model with the coupled triplet loss can be accessed at `https://youtu.be/F5kbmEuH-Gg`.

2.3 Problem Settings

Our goal is to encourage robots to learn from expert demonstrations more efficiently by taking advantage of discovering affordance-relevant distinctiveness underlying all demonstrations. To capture such distinctiveness, we could learn an attention model that predicts **affordance cues** from observations. We assume that humans have hidden affordance knowledge that is associated with visual cues. When focusing on a region of an object, humans also know what could be affordable grasps on that region. **The highlighting of such a region could serve as an affordance cue that supports the imitation learning of the learner itself.**

Our task is to pick up mugs in a way that allows pouring water in the near future. Hence, a robot could grasp a mug by reaching its gripper horizontally to the mug body, grasp the left and right sides of a handle, or grasp the front and back sides of a handle. Therefore, we have three candidate affordances (an object part and an applicable grasp): body-grasp, handle-left-right-grasp, and handle-front-back-grasp, as shown in Fig. 2.2.

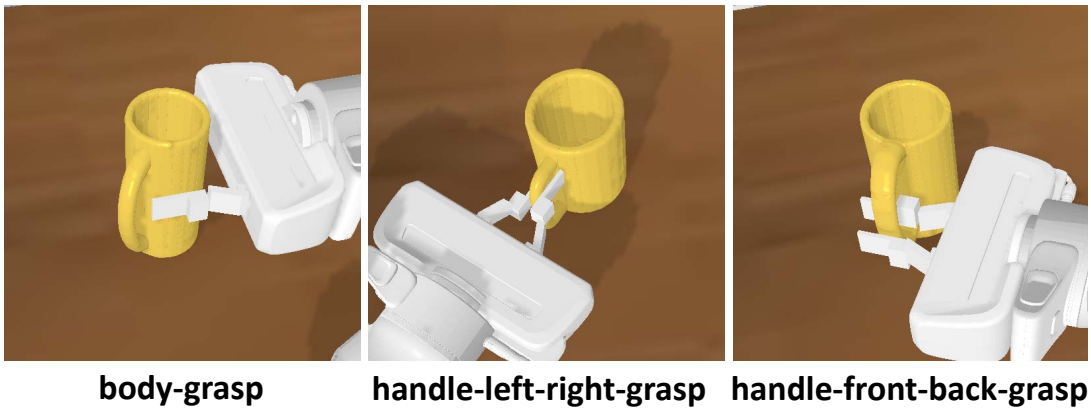


Figure 2.2: The Picture Depicts Three Examples of the Three Candidate Affordable Grasps: Body-grasp, Handle-left-right-grasp, and Handle-front-back-grasp.

We assume that robots share the same embodiment with humans: a human expert

teleoperates a robot to collect demonstration trajectories. The collected trajectories are categorized in terms of the three candidate affordances. Each trajectory τ is a sequence of triplets (o_t, s_t, a_t^*): $\tau := \{(o_t, s_t, a_t^*)\}_{t=1}^{T-1} \bigcup (s_T, o_T^*)$. T denotes the trajectory length. o_t denotes a depth image at step t. s_t denotes a state vector at t which has eight values: the relative 3-D position of the gripper to a mug, the relative 3-D Euler orientation of the gripper to mug, and two finger position values. a_t^* denotes an expert action vector that has seven values: the translation of x, y, and z; the rotation of roll, pitch, and yaw; and a value that indicates if the fingers should be closed or not. All trajectories are categorized into C **affordance categories** (C is three in this work). In each category c, there are N_c expert trajectories $\{\tau_i^c\}_{i=1}^{N_c}$.

2.4 Contrastively Learning Affordance Cues from Demonstrations

Our framework learns to reproduce humans' behavior with visual cues that hint at different affordances from human demonstrations. The learning of such visual cues is achieved by training a Siamese encoder (e.g. [129]), and the policy imitation is by a behavior-cloning-based trajectory decoder. The Siamese encoder and trajectory decoder are trained simultaneously in a contrastive learning framework.

2.4.1 A Challenge and A Graphical Model

Since human embeds her/his hidden knowledge of affordances into trajectories of different affordance categories, intuitively we could use contrastive learning to help discover affordance cues. A well-known contrastive loss is the triplet loss ([129]) defined in Equation 2.1.

$$L(\mathcal{A}, \mathcal{P}, \mathcal{N}) = \sum_{i=1}^{N} [\|f(\mathcal{A}_i) - f(\mathcal{P}_i)\|_2^2 - \|f(\mathcal{A}_i) - f(\mathcal{N}_i)\|_2^2 + M]_+ \tag{2.1}$$

where $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{N}$ denote the sets of anchor, positive, and negative trajectories; $\mathcal{A}_i$ denotes the i-th trajectory in $\mathcal{A}$ (likewise for $\mathcal{P}_i$ and $\mathcal{N}_i$); The i-th positive trajectory is sampled from the same affordance category of the i-th anchor trajectory, whereas the i-th negative trajectory is sampled from a different category. M denotes a margin value, $\|z\|_2^2$ denotes a squared Euclidean distance metric, $[z]_+$ denotes any z that is larger than zero, and, $f()$ is an encoding function that can be parameterized by a neural network.

Contrastive losses encourage a learner to discover recurring patterns in one category and discriminative patterns across different categories. From humans' perspective, shifting affordance cues would lead to the change of affordance-effect judgment such that a different action would be taken. Thus, for two trajectories that come from the same affordance category, their affordance cues would share certain similarities; for two trajectories that are sampled from different affordance categories, their affordance cues tend to be different.

However, one potential challenge in using traditional contrastive learning frameworks is that the agent might learn to exploit affordance-irrelevant information (e.g. contexts, initial configurations) to distinguish two trajectories. Inspired by earlier affordance learning from demonstration works which extract grasping preshapes ([31]) or gripper-object approaching orientations ([29]) to learn affordance clusters, we also extract **the segment of interaction state-action pairs** (shortly **interaction segment**) of each trajectory as an affordance-relevant feature. An interaction segment of a trajectory includes the state-action transitions that an actor (e.g. a gripper)

changes the state of a target object. Since affordance is about object-action-effect relationships ([78]), we use the current state and previous expert action as the state-action features per step. Specifically speaking, we extract the interaction state-action transitions from the trajectory τ: $\{(s_t, a^*_{t-1})\}^n_{t=m}|_\tau$. The curly brackets $\{\}^n_{t=m}$ mean that the state of an object changes between the step m and n due to the motion of an actor.

By contrastively learning from such interaction segments, we could learn a clean representation of affordances in the form of embeddings. Such embeddings could be named **affordance embeddings** Z^A which are computed by using Equ. 2.2. We do not directly condition robots' decision-making on Z^A. Instead, we treat Z^A as a guidance to help robots learn an attention model that extracts an affordance-cue from an observation for decision-making. To achieve this objective, we feed state s_t, image o_t, and action a^*_{t-1} at an arbitrary step t of a trajectory into a neural network that generates latent features: an affordance-cue and a processed visual feature. We then convert such latent features to **observation embedding** $z^{o,A}_t$ as illustrated in Equ. 2.3. In the rest of this paper, we use Z^A and Z^A_τ to denote the same thing: an affordance embedding for an arbitrary trajectory (τ). We use Z^A_τ when we need distinguish among different trajectories. Likewise for $z^{o,A}_t$ and $z^{o,A}_{t,\tau}$. Now we can encourage $z^{o,A}_t$ to be either closer or farther from Z^A depending on if the trajectory that gives $z^{o,A}_t$ belongs to the same affordance category of the trajectory that gives Z^A or not. This is essentially using the spatial relationships among embeddings in the space of Z^A to guide the learning of observation encoding (Equ. 2.3). This way, we link the visual processing of high-dimensional sensory input at any step of a trajectory to the affordance knowledge that can be learned faster from the lower-dimensional data of interaction segments.

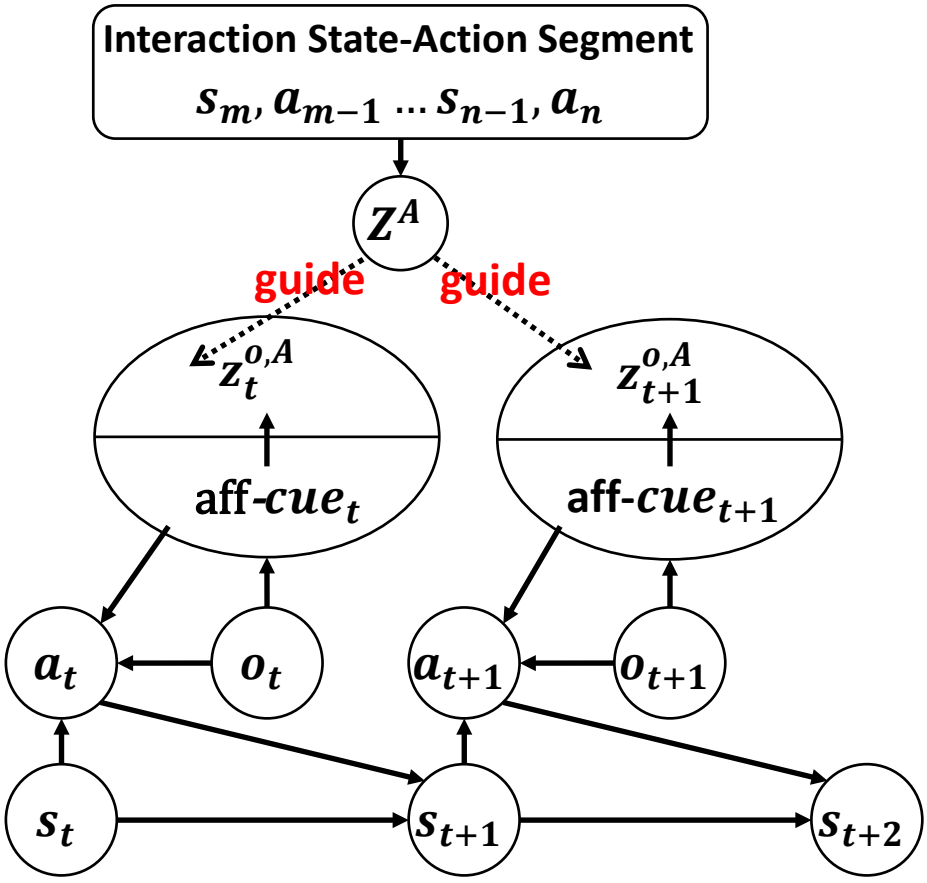


Figure 2.3: The Graphical Model of Our Siamese Encoder. Each ◯ Denotes a Node That Represents a Feature. Each Directed Edge From Node X to Node Y Represents That "Y" Depends on "X". The ⊖ Denotes Two Nodes (Features) That Are Generated by the Same Component One By One. The Dashed Edge Means That the Learning of Affordance Embedding Z_A Guides the Learning of Observation Embedding $z_t^{o,A}$ at Each Step t.

$$Z_\tau^A = f^A(\textit{interaction-segment}_\tau) = f^A(\{(s_t, a_{t-1}^*)\}_{t=m}^n|_\tau) \tag{2.2}$$

$$z_{t,\tau}^{o,A} = f^O(s_t^\tau, o_t^\tau, a_{t-1}^{\tau,*}) \tag{2.3}$$

where $f^A()$ and $f^O()$ are two encoding functions whose neural network architectures

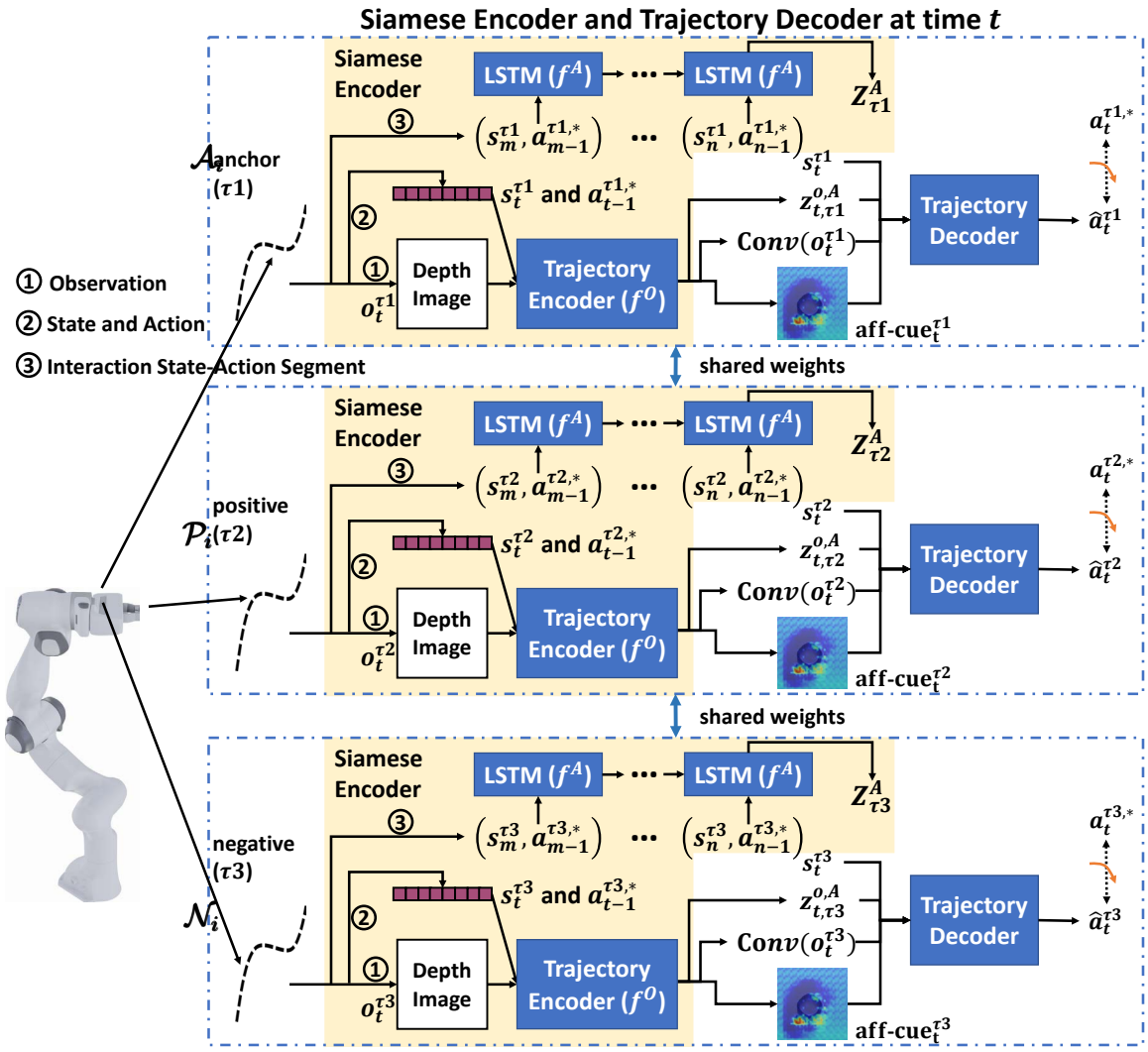


Figure 2.4: The Overall Contrastive Learning Architecture of our Siamese Encoder and Trajectory Decoder. The Dashed Edge With an Orange Arrow Means That the Distance Between its Connected Features Needs to Be Minimized.

are explained in Sec. 2.4.3; Z^A_τ encodes the interaction segment of the trajectory τ; and $z^{o,A}_{t,\tau}$ mainly encodes the high-dimension observation o_t (e.g. a depth image) at the current step t of the trajectory τ with auxiliary information like the current state s_t and previous ground-truth action a^*_{t-1}.

The graphical model that describes the above idea is illustrated in Fig. 2.3 which

shows two transitions in a trajectory. We start by extracting the interaction state-action segment of this trajectory. The interaction segment is converted to the embedding Z^A which guides the learning of $z_t^{o,A}$ as explained before. The current state s_t, observation o_t, and observation embedding $z_t^{o,A}$ determine what is the proper action a_t to take. The generation of $z_t^{o,A}$ depends on an affordance-cue (aff-cue_t) that is extracted from o_t. Note that the embedding $z_t^{o,A}$ can essentially be viewed as a non-visual part of affordance-cue. However, in this chapter, we focus on the visual part and we refer to the affordance-cue as an attention map.

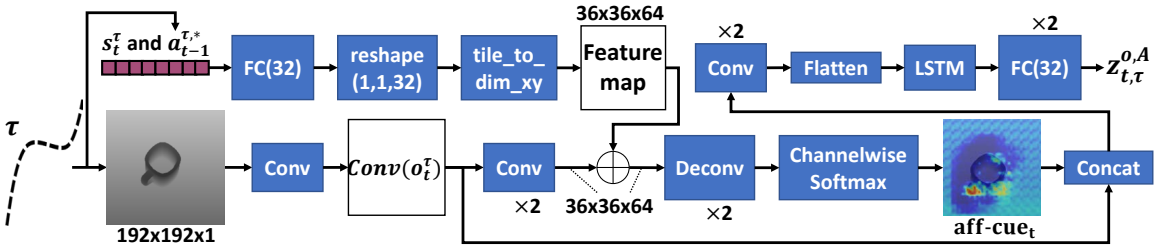


Figure 2.5: The Detailed Architecture of the Trajectory Encoder. "×#" Denotes That a Neural Network Component Needs to be Replicated # Times.

2.4.2 Siamese Encoder and Coupled Triplet Loss

Based on the graphical model, our design of Siamese encoder is depicted in the yellow region of Fig. 2.4. The Siamese encoder includes a LSTM layer to encode an interaction segment to generate Z_τ^A (Z^A for a trajectory τ). It also includes a trajectory encoder that encodes all information (image, state, and previous action) per step to generate $z_{t,\tau}^{o,A}$ ($z_t^{o,A}$ at step t in a trajectory τ). Fig. 2.4 depicts the architecture at step t, but in the training phase, we feed into the Siamese encoder a trajectory of T steps and would obtain a sequence of observation embeddings $\{z_{t,\tau}^{o,A}\}_{t=1}^T$. We also replicate a Siamese encoder into three copies and they share the same weights at any time during training. The three copies take three trajectories as inputs dur-

ing training: an anchor, positive, and negative trajectory. The anchor and positive trajectories are sampled from the same affordance category of data, while the anchor and negative trajectories come from two different categories. Given a trajectory, each copy of Siamese encoder generates Z_τ^A and $\{z_{t,\tau}^{o,A}\}_{t=1}^T$ for different possible τ as explained before. Based on Equ. 2.1, 2.2 and 2.3, we propose the coupled triplet loss (Equ. 2.4) to couple the learning of the two types of embeddings together.

$$
\begin{aligned}
L(\mathcal{A},\mathcal{P},\mathcal{N}) = \sum_{i=1}^{N}\{[\left\| Z_{\mathcal{A}_i}^A - Z_{\mathcal{P}_i}^A \right\|_2^2 - \left\| Z_{\mathcal{A}_i}^A - Z_{\mathcal{N}_i}^A \right\|_2^2 + M]_+ \\
+ \sum_{t=1}^{T}[\left\| Z_{\mathcal{A}_i}^{o,A} - z_{t,\mathcal{P}_i}^A \right\|_2^2 - \left\| Z_{\mathcal{A}_i}^{o,A} - z_{t,\mathcal{N}_i}^A \right\|_2^2 + M]_+\}
\end{aligned}
\tag{2.4}
$$

where $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{N}$ are the anchor, positive, and negative sets of demonstration trajectories; $\mathcal{A}_i$ denotes the i-th trajectory in $\mathcal{A}$ (likewise for $\mathcal{P}_i$ and $\mathcal{N}_i$); T denotes the length of an arbitrary trajectory; $f^A()$ and $f^O()$ are explained under the Equ. 2.3, which generates an affordance embedding Z^A and a observation embedding $z_t^{o,A}$ respectively. Z_τ^A (τ could be either $\mathcal{A}_i$, $\mathcal{P}_i$, or $\mathcal{N}_i$) denotes an affordance embedding Z^A for a trajectory τ; Likewise, $z_{t,\tau}^{o,A}$ denotes an observation embedding $z_t^{o,A}$ for a trajectory τ; the Sec. 2.4.3 explains $f^O()$ in more details.

The coupled triplet loss can be decomposed into two contrastive learning objectives in the two square brackets $[]_+$ that are summed together. The first objective contrastively learns affordance embedding Z^A from all extracted interaction segments. Z^A could also be learned faster due to the low-dimensionality of interaction segment data. The second objective uses Z^A to guide the learning of the observation embedding $z_t^{o,A}$. Note that $Z_{\mathcal{A}_i}^{o,A}$ is adjusted by both of the affordance embeddings $z_{t,\mathcal{P}_i}^A$ and $z_{t,\mathcal{N}_i}^A$. This way of formulating the coupled triplet loss provides a strong training signal for the observation encoder that generates $z_t^{o,A}$. In this sense, the learning of the embeddings Z^A and $z_t^{o,A}$ are coupled together.

2.4.3 The Details of Siamese Encoder

The architecture of a Siamese encoder at an arbitrary time step is illustrated in the yellow region of Fig. 2.4 and with more details in Fig. 2.5. The input can be either an anchor, positive, or negative trajectory. The definition of a trajectory is explained in Sec. 2.3.

We start by explaining the detailed formulation of the encoding function $f^A()$. At the initial step of an input trajectory, we append a dummy action vector of all zeros to provide a dummy previous action for the first step. We extract the interaction segment between the step m and n and feed them into an LSTM network ($f^A()$) to generate the affordance encoding Z^A. In our work, the values of m and n are provided by a human expert. But in reality, m and n can be determined by tracking when the relative pose of the target object to gripper starts and stops changing.

We now explain the formulation of the encoding function $f^O()$. To process the observation encoding $Z_t^{o,A}$ per step t of a whole trajectory, we feed it step-by-step into the trajectory encoder ($f^O()$) module of Siamese encoder. As depicted in Fig. 2.5, the trajectory encoder module is a two-branch architecture. The bottom branch is used to process visual features and the top branch is used to process low-dimensional features like state and action vectors. The top branch concatenates state and action together and feeds them to a fully-connected layer. After that, it tiles (repeats) the fully-connected layer feature along the x and y dimensions (e.g. 36×36) of the visual feature map generated after the third convolutional layer at the bottom branch. This way, the feature map representation of low-dimension inputs is merged with the convolutional features of visual inputs by element-wise addition. Then two deconvolution layers are used to generate a two-channel feature map. After applying the feature map with a channel-wise Softmax, its two channels represent graspable and

non-graspable probabilities per pixel respectively. We then extract the first channel of this feature map as an attention map (affordance-cue), aff-cue_t. Note that in our work, such attention maps are essentially latent attention maps because their spatial dimension matches that of the first convolution layer ($Conv(o_t^\tau)$) output, rather than the dimension of a raw input image. We then concatenate aff-cue_t with $Conv(o_t^\tau)$. After this, the two convolutional layers, one LSTM network, and two fully connected layers are used to generate $Z_t^{o,A}$. In our work we set the encoding size of $Z_t^{o,A}$ to 32.

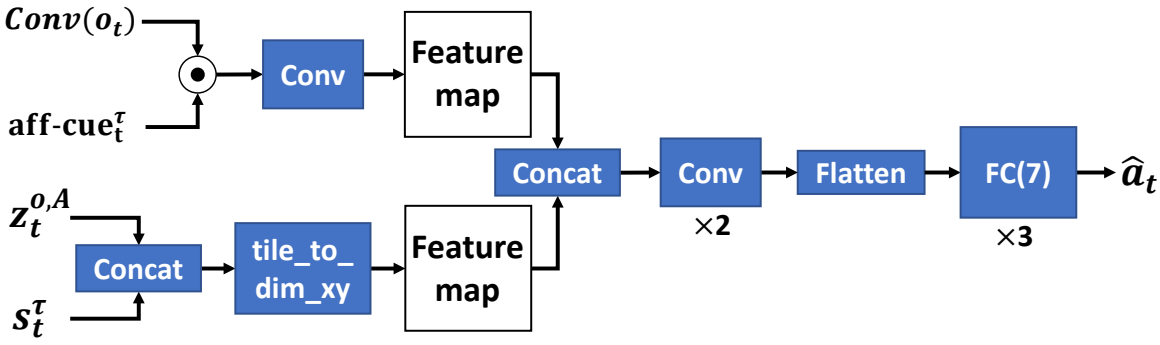


Figure 2.6: The Detailed Architecture of the Trajectory Decoder. "×#" Denotes That a Neural Network Component Needs to be Replicated # Times.

2.4.4 Policy Network as Trajectory Decoder

The design of our trajectory decoder is based on a convolutional policy network, as illustrated in Fig. 2.6. The inputs include current state s_t, the latent convolution feature $Conv(o_t)$, the affordance cue (aff-cue_t), and the contrastive embedding $z_t^{o,A}$. Since we treat a discovered affordance cue as an attention feature, we multiply aff-cue_t with each channel of the convolution feature $Conv(o_t)$.

We concatenate s_t and $z_t^{o,A}$ together and obtain a new 1-D feature. We then tile (repeat) it across x and y dimensions of the visual feature map generated by the first convolutional layer at top branch. This way, we could concatenate the visual feature

$Conv(o_t)$, state feature s_t, and contrastive embedding $z_t^{o,A}$ together along the channel dimension. This new feature is then fed into two convolution layers and three fully connected layers to obtain a predicted action $\hat{a}_t$.

The loss function for behavior decoding is based on a behavior cloning loss in Equ. 2.5. The overall loss function for training the entire Siamese encoder and trajectory decoder is a sum of the coupled triplet loss (Equ. 2.4) and behavior cloning loss (Equ. 2.5), which enables a simultaneous learning of affordance knowledge and affordance-aware grasping from expert demonstrations.

$$loss_{bc} = \sum_{i=1}^{N} \{ \sum_{t=1}^{T} [L_1(a_t^*, \hat{a}_t) + L_2(a_t^*, \hat{a}_t)] |_{\tau_i} \} \tag{2.5}$$

where L_1 and L_2 denotes L1-norm and L2-norm respectively; T denotes the length of a trajectory; $|_{\tau_i}$ denotes that the ground-truth action a_t^* is from the i-th trajectory τ_i in dataset.

2.4.5 Testing a Trained Model

When we test a trained Simease encoder and trajectory decoder (Fig. 2.7), we fix their neural network weights. Since we do not start with an entire trajectory, there is no interaction segment that provides an affordance embedding. Instead, the trained weights in Trajectory Encoder already obtained affordance knowledge due to the coupled triplet loss (Equ. 2.4). Once the trajectory decoder outputs a predicted action $\hat{a}_t$ at step t, the $\hat{a}_t$, rather than a ground-truth action as in Equ. 2.3, would be used for the computation at step $t + 1$.

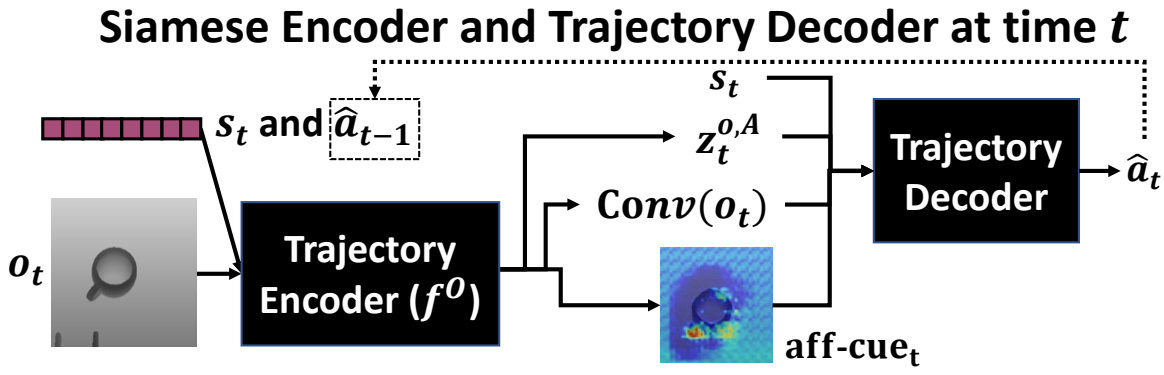


Figure 2.7: The Overall Architecture of our Siamese Encoder and Trajectory Decoder in Testing Phase. The Rectangular Filled With Black Color Means That the Model Weights Are Fixed. The Dashed Arrow and Box Represent That the Prediction of Action $\hat{a}_t$ at Each Step Would Be Fed Into the Trajectory Encoder at the Next Step.

2.5 Evaluation

We design our experiments in order to answer the following questions: 1) How helpful is the coupled triplet loss? 2) How helpful is the contrastive learning framework with a deep Siamese network? 3) How good is our model in comparison with a state-of-the-art baseline? The codes for this project can be found at [1] .

To answer 1, we compare our full model with a version that uses a normal triplet loss; to answer 2, we compare our full model with a version that does not have Siamese network and is trained totally based on behavior cloning losses; to answer 3, we compare our full model with a recently published work [90] as a baseline. Since we essentially solve an affordance-aware imitation learning problem for robotic grasping tasks, our evaluation metric involves grasping success rates which has been widely used for evaluating the learning of grasping/manipulation tasks (e.g. [62; 119]). We also show and analyze predicted affordance cues in a video from our project website[1]

[1] `https://github.com/YantianZha/Affordance-Aware-Imitation-Learning`

and partially in Fig. 2.1.

2.5.1 Evaluation Domain and Data Collection

In our evaluation domain as shown in the leftmost region of Fig. 2.1, we have a mug that is put in front of a Franka Panda Arm in the PyBullet simulator [26]. We use 24 mugs in our experiments. These mugs have different affordance characteristics and each belongs to one or more of the three affordance categories. The task for the robot is to pick up the mug and lift it up for 5 centimeters. To do this, the robot needs to intelligently infer the best way of grasping the mug, e.g., by its body, the left and right sides of its handle, or the front and back sides of its handle. This domain reflects a common service that humans may need from robots in our everyday lives. This domain is also sufficient for evaluating our algorithm due to the fact that naturally there are a large number of mugs that have different structures and geometric characteristics. It could even be necessary to consider totally different ways of grasping them from different locations.

When we collect data, we randomly select a mug model and put it in a predefined position. We fix its initial pose across all of our experiments to guarantee the existence of an affordable grasp that can be categorized into either of our three affordance categories. All demonstration trajectories are 8 steps long. We use PyBullet's function of reading users' debugging commands to interactively move the gripper to a good target pose with a mouse. Once the gripper is moved to an ideal target pose, the robot then executes with that target pose, and relevant information like observation, action, and state are recorded. We collect 27 trajectories for our training dataset. After every 10 training epochs, we test a model by randomly sampling 20 mugs and perform 20 grasps, respectively, and then record the success rate.

2.5.2 Experimental Results and Analysis

The quantitative performance is measured by grasping success rates and the qualitative performance is evaluated by showing predicted affordance cues in our video as mentioned before. The grasping results of our model, the baseline, and ablation study models are reported in Table 4.1. The success rate values are the highest testing success rates that a model achieves across all learning epochs. Table 4.1 provides experiment results of grasping success rates for four models: 1. our Siamese encoder with coupled triplet loss; 2. our Siamese network without coupled triplet loss (we apply normal triplet loss on observation embeddings); 3. our model that is not trained in a contrastive learning framework, and 4. a baseline behavior cloning work [90].

Model	Success Rate
1. Ours (Full Model): Siamese + Coupled Triplet Loss	65%
2. Ours (Ablation): Siamese + Normal Triplet Loss	35%
3. Ours (Ablation): Without Contrastive Learning	45%
4. Baseline [90]	25%

Table 2.1: The Results of Grasping Success Rates for our Model, Ablation Study Models, and a Baseline Model.

The results clearly show the advantage of using our coupled triplet loss with a Siamese neural network for the learning of affordance cues and grasping from demonstrations. An interesting finding is that the model 3 still performs better than model 2. This suggests that merely doing contrastive learning at observation level via the normal triplet loss could misguide the policy learning. Instead, the coupled triplet loss in this work contrastively learn affordance embeddings to guide the learning of observation encoding.

2.6 Related Work

2.6.1 Affordance Learning for Grasping

Regarding learning affordances for grasping, the majority of previous works use ground-truth affordance labels to learn affordances for grasping ([83; 159; 163; 139]). [83] use thermal maps to learn the graspable positions of several household objects. [159] employ transfer learning from pre-trained vision models to pixel-wise affordance prediction networks that help the robot generalize over novel objects. [163] also uses pixel-level affordances to identify multi-grasp possibilities for objects present in a cluttered area.

There are also works that propose to learn affordances from demonstrations or via imitation learning: [54; 29; 144; 31]. All of these works share similar frameworks of using classic unsupervised learning (e.g. clustering) to identify gripper control parameters that would be fed into a motion planner. In [54], the affordance is represented by contact points for grasping an object. They use a predefined tracking configuration to reduce the number of potential contact points from demonstrations. After detecting a set of contact points on a new object, nearest-neighbor classification is used to identify a template grasp that matches their demonstration data on similar objects. In [29; 144; 31], they define affordable actions in affordances as approaching angles ([29]), grasp preshape hypotheses ([144]), or grasping prototypes ([31]). Then they use clustering to find condensed representations of affordances. In this work, we leverage the expressiveness of deep neural networks to implicitly learn affordance knowledge from human demonstrations. By learning from demonstrations with a deep contrastive learning framework, we evade the need of using ground truth affordances.

2.6.2 Attention Guided Imitation Learning

Imitation learning or learning from demonstrations [90; 119] has been at the core of teaching robots to perform object-manipulation tasks in a similar way to humans performing the same task. By combining visual attention with imitation learning, robots could learn the information better by focusing on smaller but more important regions in manipulation tasks. In [2], authors use natural language descriptors that are specific to the task at hand to generate guided attention cues. The masked attention method places attention on the entire object that is to be grasped but does not focus on the specific region that the robot needs to interact with. A similar issue can be observed in [113] where the model first captures attention features of the object by generating attention maps for different stages of the object manipulation task. But the robot, in this case, can only learn to imitate the task and can not decide how to perform a specific task in a variable environment setting. Hence, one potential advantage of our approach is that it allows the robot to learn the specific graspable points in scenarios where the shape of the object (mug and its handle, in this case) can also vary restricting the possible graspable areas even for a human.

2.6.3 Discriminative Feature Learning

The objective of discriminative feature learning is to make sure that the learned features of deep neural networks can represent different inputs contrastively enough [151]. Usually, such learned features can be easily separated by k-nearest neighbors algorithms [40]. Various approaches have been proposed to address the discriminative learning problem for deep neural networks. One popular approach is Siamese neural network [15] that was proposed in 1994 for verifying signatures. Quite a few works used Siamese neural networks as a backbone for new deep network models of discrim-

inative feature learning. Siamese neural network can be combined with ConvNets and trained with a Binary Cross Entropy loss as in [63], or triplet loss as in [129]). Recently deep learning based Siamese neural networks are also applied in many new applications, like face recognition ([129; 135]), object discovery ([138; 55]), object co-segmentation ([10; 79; 92]), and re-identification as in [169; 97].

Besides using Siamese neural networks for discriminative feature learning, [151] proposes to replace normal classification loss functions in ConvNets with the Center loss. Center loss works by minimizing the intra-class variations and meanwhile keeping the inter-class feature variations separable enough. The work [155] proposes a prototype-based discriminative feature learning (PDFL) method. The work [74] follows a similar idea to [151] and designs a discriminative feature learning algorithm for domain adaption.

2.7 Concluding Remarks

We present an imitation learning algorithm that seeks training guidance not only from teachers' actions but from simultaneously discovering teachers' hidden affordance bias as well. We propose a contrastive learning framework with a Siamese encoder for affordance discovery and a trajectory decoder for policy learning. We represent affordance cues as visual attention. We further propose the coupled triplet loss to encourage the learned discriminative features to be more affordance-relevant. To the best of our knowledge, we initialize the direction of bridging the gap between affordance discovery and policy engineering/learning by achieving the two objectives together via an end-to-end deep neural network. Our work inherits the benefits of the class of works that learns affordances from demonstrations: there is no need of collecting ground-truth affordance labels for each image or pixel. However, such works focus on affordance learning and predicted affordances is still used by external mo-

tion planners. Our evaluation shows that our algorithm achieves the highest grasping success rate and predicts meaningful affordance cues.

While the affordance-cue learning made the perception more "conscious", its learning is "model-free" that does not support "long-term thinking". Therefore, in Chapter 3, I would introduce my works that learn neural networks to couple planning and perception.

2.7.1 Significance

We believe our affordance-guided policy imitation framework has potential for solving more complex tasks, which could be pursued in the future extensions of this work:

1) The tasks that involve multiple levels of interactions, instead of only the interaction between gripper and mug. Most tool-using tasks would require multiple levels of interactions;

2) The prediction of affordance-cue could be conditioned on different high-level tasks such that the attention map would be different even on the same object but w.r.t different tasks. In our work, we have a fixed high-level task of picking-to-pour-water, but if there is another task like pick-and-relocate, the discovered affordance cues might be different. This can be achieved by integrating our work into a hierarchical imitation learning framework.

3) It might also be interesting to explore how the coupled triplet loss could also be used to address other robot learning problems other than affordance-aware policy imitation.

Chapter 3

LEARNING A SHALLOW PLANNING MODEL FOR PLAN RECOGNITION

3.1 Inspirations from Neural Language Models

Traditional symbolic planning approaches usually require a full model with complete definitions of actions (with preconditions and effects). Such full models are usually defined by STRIPS ([34]) or PDDL ([3]). To enable a cognitive learning agent with planning functions, it is crucial to research a learning counterpart of classical planning algorithms. The benefits are two-fold: 1) learning-to-plan allows the planning to be more robust and better adapt to specific environments and tasks without requiring a human domain expert to make domain modifications; 2) learning-to-plan allows a planning component to better match other cognitive components like perception by simultaneous learning them together.

Neural Language Models refer to the class of deep learning methods for learning language models. Such language models could solve tasks like translation, language embedding, and language generation. By viewing symbolic plan traces as sentences and Neural Language Models as approximated planners, we can potentially adopt some Neural Language Models to address planning tasks.

In this chapter, I mostly focus on symbolic plan recognition tasks that can be addressed by learning a shallow planning model. There is a close connection between planning and plan recognition: Solving plan recognition problems can be reduced to solving planning problems, as illustrated and proven in [117; 114; 133]. However, these plan recognition works rely on fully defined planning models. In the rest of the sections of this chapter, I will draw a connection between Neural Language Models

and Planning. That is to say, instead of constructing a formal planning model, we could learn a shallow planning model from plan corpora with novel algorithms inspired by Neural Language Models. The various incompleteness of shallow planning models and other types of planning models are summarized in Fig. 3.1.

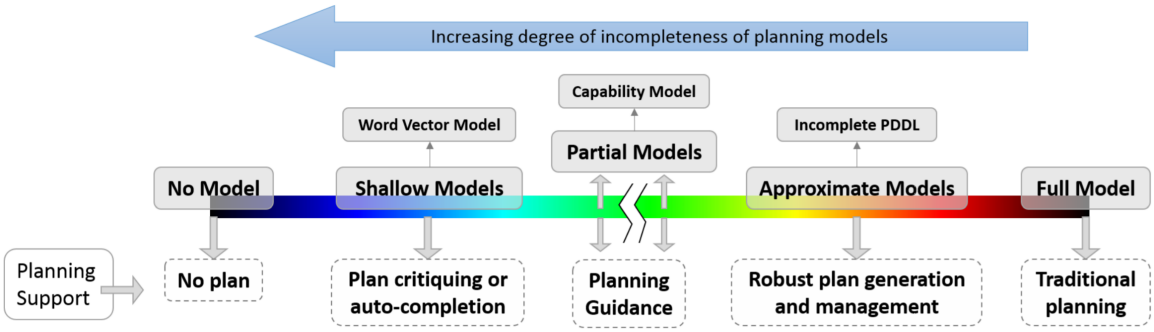


Figure 3.1: Schematic View of Incomplete Models and Their Relationships in the Spectrum of Incompleteness (from [176])

3.1.1 Chapter Highlights

- In Sec. 3.2, I will start by discussing two methods in [176] to learn shallow planning models via Neural Language Models for solving plan recognition problems. However, they rely on optimal visual recognition of actions from videos.

- In Sec. 3.3, I introduce an extension work [166] that could handle erroneous perception of actions from videos by directly learning from observed (uncertain) action distributions.

3.1.2 Background

Word2Vec and Hierarchical Softmax for Learning Distributed Representations of Actions

In this section we make a brief review about the Skip-gram `Word2Vec` works based on [86]. To be consistent with the whole paper, we use the term "action" and treat it an equivalence to the term "word" in other papers which introduces `Word2Vec`. The `Word2Vec` works such as Skip-gram models can be used as fundations for learning distributed representations of actions (action embeddings). We then explain how the learned action embeddings could be used for plan recognition as introduced in [146].

Given a corpora which contains action (word) sequences, a Skip-gram model is trained by maximizing the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-\mathcal{W}\leq j\leq\mathcal{W},j\neq 0}\log p(a_{t+j}|a_t) \tag{3.1}$$

where T is the length of a sequence, $\mathcal{W}$ is the context window size, a_t is fed as the model input, a_I, and a_{t+j} is used as the target action (word), a_O. The probability $p(a_O|a_I)$ is computed as:

$$p(a_O|a_I) = \frac{exp({v'}_{a_O}^{T} v_{a_I})}{\sum_{a=1}^{A} exp({v'}_{a}^{T} v_{a_I})} \tag{3.2}$$

which is essentially a softmax function. A denotes a vocabulary of all possible actions. Other symbols follows the definition in Equation 3.1. We can use this equation that computes $p(a_O|a_I)$ to compute $p(a_{t+j}|a_t)$ in Equation 3.1.

The $p(a_{t+j}|a_t)$ in the `Word2Vec` with hierarchical softmax is calculated in the following manner. The output layer's weight matrix of the regular `Word2Vec` is replaced

by a binary tree whose leaf nodes are words in the trained vocabulary. Every node on the path from the root node until the leaf node has a vector, excluding the leaf node. The input action a_i is converted into an embedding h which is the input into the binary tree component. The probability of this input vector h that could go to a particular leaf node is calculated by following the path from the root node to the target leaf node, using the following formula:

$$p(a_{t+j}|a_t) = \prod_{i=1}^{L(a_{t+j})-1} \Big\{\sigma(\mathbb{I}(n(a_{t+j}, i+1) = child(n(a_{t+j}, i))) \cdot v_{n(a_{t+j},i)} \cdot h)\Big\}, \tag{3.3}$$

where $\mathbb{I}(x)$ is a function that returns 1 if the next node on the path to the target leaf node is on the left of the current node, and -1 if the next node is to the right. $L(a_{t+j})$ is the length of path from root to the leaf node a_{t+j}, $v_{n(a_{t+j},i)}$ is the vector of the i-th node along the path. h is the embedding obtained by multiplying the embedding matrix and vector of the input action a_t. h is the vector that represents the input action in the embedding space. In order to maximize the probability, the vectors of the intermediary nodes are updated with each training sample which has the target action a_t and an action in its context a_{t+j}.

Recurrent Neural Networks (RNNs)

In this chapter, we consider a specific type of RNNs, which generates an output sequence with the same length of an input sequence. Note that in natural language processing (NLP) community, various RNN models, such as deep RNNs [105], Bi-LSTM-CRF [5], etc., were designed to suit different complicated NLP tasks. The more complex the model is, the more data are required to build the model. Compared to NLP tasks, our plan recognition task is relatively not that complicated (plans in

the plan library are formally represented by action sequences, while natural language data in NLP tasks are often unstructured and ambiguous). In addition, our data is relatively smaller than that used in many RNN models in complicated NLP tasks. To avoid overfitting, we chose a basic RNN model (with a relatively small number of parameters). Given an input sequence $\mathbf{x}$, an RNN model could predict an output sequence $\mathbf{y}$, in which output y_t at each step depends on the input x_t at that step, and the hidden state at the previous step. If we unroll this RNN cell along T steps, it could accept an input sequence $\mathbf{x} = (x_1..., x_T)$, and compute a sequence of hidden states $\mathbf{h} = (h_1, h_2, ..., h_T)$, as shown in Fig. 3.2. For each of these hidden states h_t $(1 \leq t \leq T)$, it contributes to predicting the next step output y_{t+1}, and thus RNN computes an output vector sequence $\mathbf{y} = (y_1, ..., y_T)$, by iterating the following equations from $t = 1$ to T:

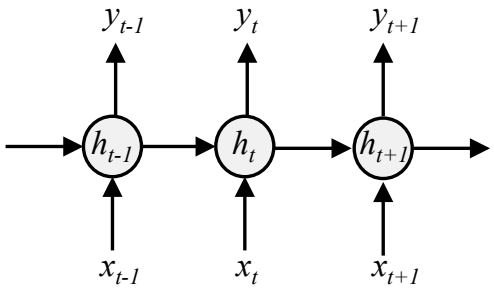


Figure 3.2: The Framework of Recurrent Neural Networks

$$h_t = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{3.4}$$

$$y_t = W_{hy}h_t + b_y \tag{3.5}$$

where the W terms denote weight matrices, i.e., W_{xh} is the input-hidden weight matrix, W_{hh} is the hidden-hidden weight matrix, and W_{hy} is the hidden-output matrix. The b terms denote bias vectors, i.e., b_h is hidden bias vector, and b_y is the output bias

vector. $\mathcal{H}$ is the hidden layer function, which is usually an elementwise application of a sigmoid function. The RNN could also be utilized to directly generate, in principle, infinitely long future outputs, given a single input x_t. The sequence of future outputs could be generated by directly feeding the output y_t at a step t, to the input x_{t+1} at the next step $t+1$, by assuming what it predicts is reliable (i.e., $y_t = x_{t+1}$). As for training the RNN as a sequence generation model, we could utilize y_t to parameterize a predictive distribution $P(x_{t+1}|y_t)$ over all of the possible next inputs x_{t+1}, by minimizing the loss:

$$\mathcal{L}(\mathbf{x}) = -\sum_{t=1}^{T} logP(x_{t+1}|y_t). \tag{3.6}$$

where T is the number of steps of an observed plan trace, x_{t+1} is the observed action at step $t+1$, and y_t is the output at step t as well as the prediction of what would happen at step $t+1$. To estimate y_t based on $x_1, \ldots, x_t$, we exploit the Long Short-term Memory (LSTM) model [46; 130], which has been demonstrated effective on generating sequences, to leverage long term information prior to x_t and predict y_t based on current input x_t. We can thus rewrite Equation (3.6) as:

$$\mathcal{L}(\mathbf{x};\theta) = -\sum_{t=1}^{T} logP(x_{t+1}|y_t)\text{LSTM}(y_t|x_{1:t};\theta), \tag{3.7}$$

where $\text{LSTM}(y_t|x_{1:t};\theta)$ indicates the LSTM model estimates y_t based on current input x_t and memories of previous input prior to x_t. θ are parameters including W_{xh}, W_{hh}, W_{hy}, b_h, and b_y.

The framework of LSTM [46; 130] is shown in Fig. 3.3, where x_t is the tth input, h_t is the tth hidden state. i_t, f_t, o_t and c_t are the tth *input gate*, *forget gate*, *output gate*, *cell* and *cell input* activation vectors, respectively, whose dimensions are the same as the hidden vector h_t. LSTM is implemented by the following functions:

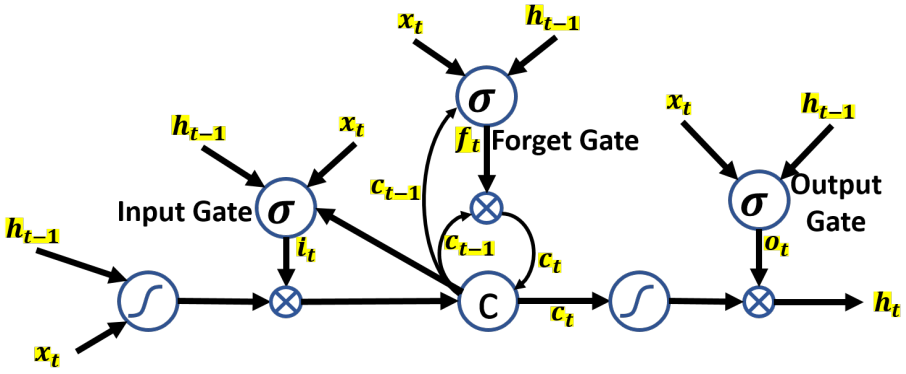


Figure 3.3: The Framework of the Long Short-term Memory (LSTM) Cell

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3.8}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3.9}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.10}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{3.11}$$

$$h_t = o_t \circ \tanh(c_t) \tag{3.12}$$

where $\circ$ indicates the Hadamard product, σ is the logistic sigmoid function, W_{xi} is an input-input gate matrix, W_{hi} is a hidden-input gate matrix, W_{ci} is a cell-input gate matrix, W_{xf} is an input-forget gate matrix, W_{hf} is a hidden-forget gate matrix, W_{cf} is a cell-forget gate matrix, W_{xc} is an input-cell gate matrix, W_{hc} is a hidden-cell gate matrix, W_{xo} is an input-output gate matrix, W_{ho} is a hidden-output gate matrix, W_{co} is a cell-output gate matrix. b_i, b_f, b_c, and b_o are bias terms. Note that the matrices from cell to gate vectors (i.e., W_{ci}, W_{cf} and W_{co}) are diagonal, such that each element e in each gate vector only receives input of element e of the cell vector. The major innovation of LSTM is its memory cell c_t which essentially acts as an accumulator of the state information. c_t is accessed, written and cleared by self-parameterized controlling gates, i.e., *input, forget, output* gates. Each time a new input x_t comes, its information is accumulated to the memory cell if the input gate i_t is activated.

The past cell status c_{t-1} could be forgotten in this process if the forget gate f_t is activated. Whether the latest cell output c_t is propagated to the final state h_t is further controlled by the output gate o_t. The benefit of using the memory cell and gates to control information flow is the gradient is trapped in the cell and prevented from vanishing too quickly.

3.2 Recognizing Plans with Learned Shallow Planning Models

With a fully defined planning model, we can optimally predict what will be the next state given the current state and action. For symbolic planning models, a full model would require a complete symbolic definition of actions with their preconditions and effects. However, to make a planning and perception model adaptable so that they can work with each other as a whole, it would be promising to model them as parametric models like neural networks. In contrast to full planning models, shallow planning models refer to models that can be used to approximately solve planning tasks yet without complete and syntactic definitions of planning domain knowledge. In [145; 176], shallow planning models are approximated by a Word2Vec model ([123]) trained on corpora of plan traces.

There are certain similarities between planning and natural language processing (NLP) tasks. The learning of a Neural Language Model requires natural language sentences and each sentence is formed by a sequence of words. Likewise, a plan trace can be viewed as a sentence if we treat each action in the plan trace as a word. Then the dataset of natural language sentences for learning an NLP model helps in a similar way to using a plan library for learning a shallow planning model.

Example: A plan library $\mathcal{L}$ in the *blocks* domain is assumed to have four plans as shown below:

plan 1: *pick-up-B stack-B-A pick-up-D stack-D-C*
plan 2: *unstack-B-A put-down-B unstack-D-C put-down-D*
plan 3: *pick-up-B stack-B-A pick-up-C stack-C-B pick-up-D stack-D-C*
plan 4: *unstack-D-C put-down-D unstack-C-B put-down-C unstack-B-A put-down-B*

An observation $\mathcal{O}$ of action sequence is shown below:

observation: *pick-up-B* ϕ *unstack-D-C put-down-D* ϕ *stack-C-B* ϕ ϕ

Given the above input, our `DUP` algorithm outputs plans as follows:

pick-up-B stack-B-A unstack-D-C put-down-D pick-up-C stack-C-B pick-up-D stack-D-C

3.2.1 Problem Setting

In the work [176], we learn shallow planning models from such a plan library for solving plan recognition problems. A plan library, denoted by $\mathcal{L}$, is composed of a set of plans $\{p\}$, where p is a sequence of actions, i.e., $p = \langle a_1, a_2, \ldots, a_n \rangle$ where a_i, $1 \leq i \leq n$, is an action name (without any parameter) represented by a string. For example, a string *unstack-A-B* is an action meaning that *a robot unstacks block A from block B*. We denote the set of all possible actions by $\bar{\mathcal{A}}$ which is assumed to be known beforehand. For ease of presentation, we assume that there is an empty action, ϕ, indicating an unknown or not observed action, i.e., $\mathcal{A} = \bar{\mathcal{A}} \cup \{\phi\}$. An observation of an *unknown* plan $\tilde{p}$ is denoted by $\mathcal{O} = \langle o_1, o_2, \ldots, o_M \rangle$, where $o_i \in \mathcal{A}$, $1 \leq i \leq M$, is either an action in $\bar{\mathcal{A}}$ or an empty action ϕ indicating the corresponding action is missing or not observed. Note that $\tilde{p}$ is not necessarily in the plan library $\mathcal{L}$, which makes the plan recognition problem more challenging, since matching the observation to the plan library will not work any more.

We assume that the human is making a plan of at most length M. We also assume that at any given point, the planner is able to observe $M - k$ of these actions. The k unobserved actions might either be in the suffiix of the plan, or in the middle. Our aim is to suggest, for each of the k unobserved actions, l possible choices from which

the user can select the action. (Note that we would like to keep l small, ideally close to 1, so as not to overwhelm users). Accordingly, we will evaluate the effectiveness of the decision support in terms of whether or not the user's best/intended action is within the suggested l actions.

Specifically, our recognition problem can be represented by a triple $\Re = (\mathcal{L}, \mathcal{O}, \mathcal{A})$. The solution to $\Re$ is to discover the unknown plan $\tilde{p}$, which is a plan with unknown observations, that best explains $\mathcal{O}$ given $\mathcal{L}$ and $\mathcal{A}$. We have the following assumptions **A1-A3**:

A1: The length of the underlying plan to be discovered is known, which releases us from searching unlimited length of plans.

A2: The positions of missing actions in the underlying plan is known in advance, which releases us from searching missing actions in between observed actions.

A3: All actions observed are assumed to be correct, which indicates there is no need to criticize or rectify the observed actions.

Note that assumptions **A1** and **A2** can be removed by leveraging auxiliary knowledge, such as domain models, as done by previous approaches [115; 33; 134]. Building such auxiliary knowledge often requires a lot of human efforts, we thus assume we do not have that knowledge by considering assumptions **A1** and **A2**. The Fig. 3.4 shows an example for each of L, O, and R.

3.2.2 DUP – Discovering Underlying Plans

We introduce two approaches of learning shallow planning models via NLP methods as illustrated in Fig. 3.5. One approach is named DUP (**D**iscovering **U**nderlying **P**lans) that leans semantic correlations among actions, which can be treated as words, for plan recognition. The semantic correlations are captured by learning a Word2Vec

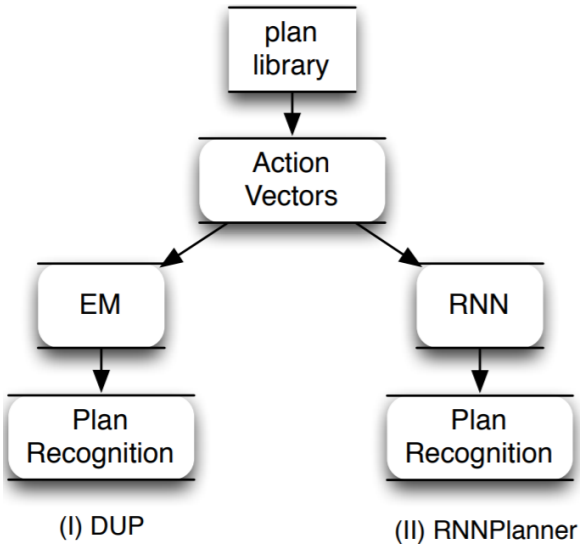


Figure 3.5: Learning Shallow Planning Models via Neural Language Models: DUP and RNNPlanner (from [176]).

model on the plan library L. With a learned action (or word) embedding model, we could do planning or plan recognition by iteratively finding actions that have the closest embedding distances to observed actions (a greedy algorithm) or by a more efficient Expectation-Maximization searching algorithm.

Our `DUP` approach to the recognition problem $\Re$ functions by two phases. We first learn vector representations of actions using the plan library $\mathcal{L}$. We then iteratively sample actions for unobserved actions o_i by maximizing the probability of the unknown plan $\tilde{p}$ via the EM framework. We present `DUP` in detail in the following subsections.

Maximizing Probability of Unknown Plans

With the vector representations learnt in the last subsection, a straightforward way to discover the unknown plan $\tilde{p}$ is to explore all possible actions in $\bar{\mathcal{A}}$ such that $\tilde{p}$ has

the highest probability, which can be defined similar to Equation (3.1), i.e.,

$$\mathcal{F}(\tilde{p}) = \sum_{k=1}^{M} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{k+j}|w_k) \tag{3.13}$$

where w_k denotes the kth action of $\tilde{p}$ and M is the length of $\tilde{p}$. As we can see, this approach is exponentially hard with respect to the size of $\bar{\mathcal{A}}$ and number of unobserved actions. We thus design an approximate approach in the Expectation-Maximization framework to estimate an unknown plan $\tilde{p}$ that best explains the observation $\mathcal{O}$.

To do this, we introduce new parameters to capture "weights" of values for each unobserved action. Specifically speaking, assuming there are X unobserved actions in $\mathcal{O}$, i.e., the number of ϕs in $\mathcal{O}$ is X, we denote these unobserved actions by $\bar{a}_1, ..., \bar{a}_x, ..., \bar{a}_X$, where the indices indicate the order they appear in $\mathcal{O}$. Note that each $\bar{a}_x$ can be any action in $\bar{\mathcal{A}}$. We associate each possible value of $\bar{a}_x$ with a weight, denoted by $\bar{\Gamma}_{\bar{a}_x,x}$. $\bar{\Gamma}$ is a $|\bar{\mathcal{A}}| \times X$ matrix, satisfying

$$\sum_{o \in \bar{\mathcal{A}}} \bar{\Gamma}_{o,x} = 1,$$

where $\bar{\Gamma}_{o,x} \geq 0$ for each x. For the ease of specification, we extend $\bar{\Gamma}$ to a bigger matrix with a size of $|\bar{\mathcal{A}}| \times M$, denoted by Γ, such that $\Gamma_{o,y} = \bar{\Gamma}_{o,x}$ if y is the index of the xth unobserved action in $\mathcal{O}$, for all $o \in \bar{\mathcal{A}}$; otherwise, $\Gamma_{o,y} = 1$ and $\Gamma_{o',y} = 0$ for all $o' \in \bar{\mathcal{A}} \wedge o' \neq o$. Our intuition is to estimate the unknown plan $\tilde{p}$ by selecting actions with the highest weights. We thus introduce the weights to Equation (3.2), as shown below,

$$p(w_{k+j}|w_k) = \prod_{i=1}^{L(w_{k+j})-1} \Big\{ \sigma(\mathbb{I}(n(w_{k+j}, i+1) = child(n(w_{k+j}, i))) \cdot av_{n(w_{k+j},i)} \cdot bv_{w_k}) \Big\}, \tag{3.14}$$

where $a = \Gamma_{w_{k+j},k+j}$ and $b = \Gamma_{w_k,k}$. We can see that the impact of w_{k+j} and w_k is penalized by weights a and b if they are unobserved actions, and stays unchanged, otherwise (since both a and b equal to 1 if they are observed actions).

We assume $X \sim Multinomial(\Gamma_{\cdot,x})$, i.e., $p(X = o) = \Gamma_{o,x}$, where $\Gamma_{o,x} \geq 0$ and $\sum_{a \in \bar{\mathcal{A}}} \eta^a = 1$. $P(\tilde{p}|\Gamma) = \prod_x \Gamma_{o,x}$

We redefine the objective function as shown below,

$$\mathcal{F}(\tilde{p}, \Gamma) = \sum_{k=1}^{M} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{k+j}|w_k), \tag{3.15}$$

where $p(w_{k+j}|w_k)$ is defined by Equation (3.14). The gradient of Equation 3.15 is shown below,

$$\frac{\partial \mathcal{F}}{\partial \Gamma_{o,x}} = \frac{4c(L(o) - 1)}{\Gamma_{o,x}}. \tag{3.16}$$

The only parameters needed to be updated are Γ, which can be easily done by gradient descent, as shown below,

$$\Gamma_{o,x} = \Gamma_{o,x} + \delta \frac{\partial \mathcal{F}}{\partial \Gamma_{o,x}}, \tag{3.17}$$

if x is the index of unobserved action in $\mathcal{O}$; otherwise, $\Gamma_{o,x}$ stays unchanged, i.e., $\Gamma_{o,x} = 1$. Note that δ is a learning constant.

With Equation (3.17), we can design an EM algorithm by repeatedly sampling an unknown plan according to Γ and updating Γ based on Equation (3.17) until reaching convergence (e.g., a constant number of repetitions is reached).

Overview of our DUP approach

An overview of our DUP algorithm is shown in Algorithm 1. In Step 2 of Algorithm 1, we initialize $\Gamma_{o,k} = 1/M$ for all $o \in \bar{\mathcal{A}}$, if k is an index of unobserved actions in $\mathcal{O}$; and otherwise, $\Gamma_{o,k} = 1$ and $\Gamma_{o',k} = 0$ for all $o' \in \bar{\mathcal{A}} \wedge o' \neq o$. In Step 4, we view $\Gamma_{\cdot,k}$ as a probability distribution, and sample an action from $\bar{\mathcal{A}}$ based on $\Gamma_{\cdot,k}$ if k is an unobserved action index in $\mathcal{O}$. In Step 5, we only update $\Gamma_{\cdot,k}$ where k is an unobserved action index. In Step 6, we linearly project all elements of the updated Γ to between 0 and 1, such that we can do sampling directly based on Γ in Step 4.

In Step 8, we simply select $\bar{a}_x$ based on

$$\bar{a}_x = \arg\max_{o \in \bar{\mathcal{A}}} \Gamma_{o,x},$$

for all unobserved action index x.

Algorithm 1 Framework of our DUP algorithm

Input: plan library $\mathcal{L}$, an observation $\mathcal{O}$

Output: plan $\tilde{p}$

1: learn vector representation of actions based on $\mathcal{L}$
2: initialize $\Gamma_{o,k}$ with $1/M$ for each $o \in \bar{\mathcal{A}}$ and each k that is an unobserved action index
3: **while** the maximal number of repetitions Λ is not reached **do**
4: sample unobserved actions in $\mathcal{O}$ based on Γ
5: for each unobserved action x, update $\Gamma_{\cdot,x}$ based on Equation (3.17)
6: project Γ to [0,1]
7: **end while**
8: select actions for unobserved actions with the largest weights in Γ
9: **return** $\tilde{p}$

Our DUP algorithm framework belongs to a family of policy gradient algorithms, which have been successfully applied to complex problems, e.g., robot control [99], natural language processing [12]. Our formulation is unique in how it recognizes plans, in comparison to the existing methods in the planning community. The first step of Algorithm 1 is $O(Iters \times T_{max} \times |\mathcal{L}| \times WinSize)$, where $Iters$ is the maximal number of iterations for vector representations of actions, and T_{max} is the maximal length of plans in $\mathcal{L}$ and $|\mathcal{L}|$ is the number of plans in $\mathcal{L}$, and $WinSize$ is the size of the window ($WinSize = 2c+1$). The time cost of Step 2 is $O(|\mathcal{A}| \times unobserved(\mathcal{O})$, and the time cost of Steps 2 to 7 is $O(\Lambda \times |\mathcal{A}| \times unobserved(\mathcal{O})$, where $unobserved(\mathcal{O})$ is the number

of unobserved actions in $\mathcal{O}$. In general, the time cost of the first step is much higher than other steps, i.e., the time cost of Algorithm 1 is $O(Iters \times T_{max} \times |\mathcal{L}| \times WinSize)$.

3.2.3 RNNPlanner

The other approach is named as RNNPlanner (**R**ecurrent **N**eural **N**etwork based **P**lanner). RNNPlanner works by learning a recurrent neural network (a Long-Short-term Memory net) as a sequential generative model to directly output missing or future actions from an observed plan prefix. As such, planning or plan recognition tasks can be accomplished. As we model our plan recognition problem as an action-sequence generation problem, our aim of exploring RNN-LSTM architecture is to leverage longer-horizon of actions to help improve the accuracy of generating new actions based on previously observed or generated actions. We will first introduce the RNN-LSTM architecture, and then introduce our `RNNPlanner` model.

Discovering Underlying Plans with the RNN Model

With the distributed representations of actions addressed in Section 3.1.2, we view each plan in the plan library as a sequence of actions, and the plan library as a set of action sequences, which can be utilized to train the RNN model. The framework of RNN with sequences of actions can be seen from Fig. 3.6, which is similar to [47]. The bottom row in Fig. 3.6 is an input action sequence "pick-up-B, stack-B-A, pick-up-C, stack-C-B, pick-up-D". The "Embedding Layer" computes vector representations of actions, which is pretrained by Section 3. Similar to a classic RNN cell, the LSTM cell feeds its output to both itself as a hidden state, and the softmax layer to obtain a probability distribution over all actions in the action vocabulary $\mathcal{A}$. From the perspective of the LSTM cell at the next step, it receives a hidden state from the previous step h_{t-1}, an action vector at the current step x_t. To obtain the index of

most possible action, our model samples over the action distribution output from softmax layer. That retrieved index could be mapped to an action in the vocabulary $\bar{\mathcal{A}}$.

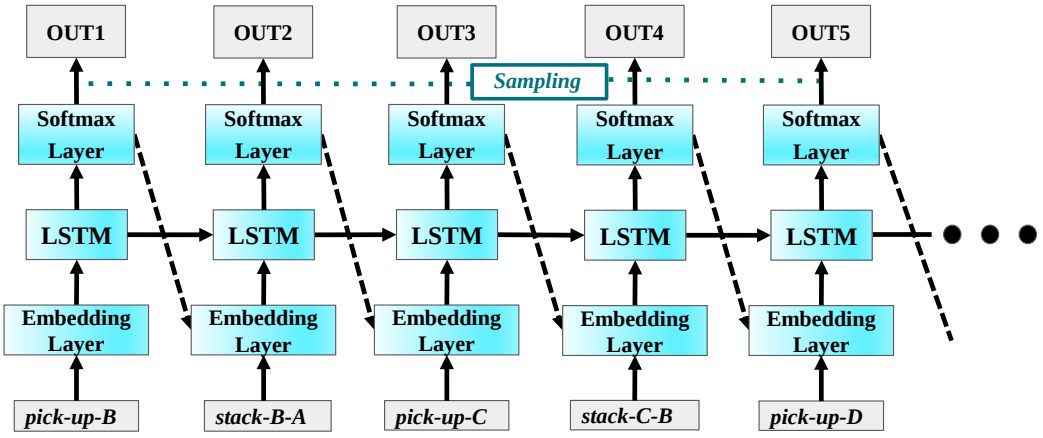


Figure 3.6: The Framework of our RNNPlanner Approach

The top row in Fig. 3.6 is the output sequence, denoted by "OUT1, OUT2, OUT3, OUT4, OUT5, ...", which corresponds to the estimated sequence "y_1, y_2, y_3, y_4, y_5, ..." in Equation (3.18). Note that we exploit the dotted arrow to indicate two folds of meanings in Fig. 3.6. When training the RNN model, the one pointed by the head of the dotted arrow (the embedding of the input) is used to compute the cross entropy with the output at tail (output of LSTM cell at the previous step), and next-step observation as the input at the head, to train the model. When using the trained RNN model to discover unknown actions, the model assumes what it predicts is the real next input, and takes it to continue its prediction. Thus the one pointed by the head is copied and identical to the one denoted by the tail. For example, the embedding of "stack-B-A" is copied from the prediction vector of "OUT1" if the input "stack-B-A" was unknown. In addition, the arrows between each of two LSTM cells shows the unrolling of a LSTM cell. The horizontal dashed line suggests that we obtain the action output at each step, by sampling from probability distribution, provided by

the softmax layer. With the trained RNN model, we can discover underlying actions by simply exploiting the RNN model to generate unknown actions based on observed or already discovered actions.

An overview of our `RNNPlanner` algorithm can be seen from Algorithm 2. In Algorithm 2, Step 1 is the same as Step 1 of Algorithm 1. In Step 2, we let $y_t = x_{t+1}$, such that, given $p(x_{t+1}|y_t) = 1$, Equation (3.18) can be calculated by

$$\mathcal{L}(\mathbf{x};\theta) = -\sum_{t=1}^{T} log\text{LSTM}(x_{t+1}|x_{1:t};\theta), \tag{3.18}$$

where $\mathbf{x}$ is a plan in $\mathcal{L}$. We use MLE (Maximal Likelihood Estimation) to estimate parameters θ given $x_{1:t}$ as input and x_{t+1} as output for each $t \in [1, |\mathbf{x}|)$, where $\mathbf{x}$ is a plan from $\mathcal{L}$.

For example, given the observation:

pick-up-B ϕ *unstack-D-C put-down-D* ϕ *stack-C-B* ϕ ϕ

we can generate the first ϕ based on *pick-up-B*, the second ϕ based on actions from *pick-up-B* to *put-down-D*, the third ϕ based on all previous actions (including the generated actions for ϕs).

The running time complexity of Step 1 in Algorithm 2 is the same as addressed in the Algorithm 1, i.e., $O(Iters \times T_{max} \times |\mathcal{L}| \times WinSize)$. The time cost of Steps 2 to 7 of Algorithm 2 is $O(\Lambda \times |\mathcal{L}| \times T_{max}^2)$, where T_{max}^2 is the time cost of updating θ (Step 5) based on plan p since it needs to scan each action a in p and a's preceding actions when computing the objective $\mathcal{L}(\mathbf{x};\theta)$ as shown in Equation 3.18. The time cost of Steps 8 to 15 is $O(|\mathcal{O}|^2)$ based on the parameters θ learnt by previous steps. In summary, the complexity of Algorithm 2 is $O(\Lambda \times |\mathcal{L}| \times T_{max}^2)$, considering T_{max} is much larger than $WinSize$, and the length of $\mathcal{O}$ is general much less than $|\mathcal{L}| \times T_{max}^2$.

Algorithm 2 Framework of our `RNNPlanner` algorithm

Input: plan library $\mathcal{L}$, observed actions $\mathcal{O}$

Output: plan $\tilde{p}$

learn vector representation of actions based on $\mathcal{L}$
initialize θ with random values
while the maximal number of repetitions Λ is not reached **do**
    for each plan $p \in \mathcal{L}$ **do**
        update parameters θ by optimizing the objective $\mathcal{L}(\mathbf{x}; \theta)$ based on the vector representations of p
    end for
end while
let $\tilde{p} = NULL$
for each $t = 1$ to $|\mathcal{O}|$ **do**
    if $o_t \in \mathcal{O}$ is ϕ **then**
        generate y_t based on $LSTM(y_t|\tilde{p}; \theta)$
        $o_t = y_t$
    end if
    $\tilde{p} = [\tilde{p}|o_t]$
end for
return $\tilde{p}$

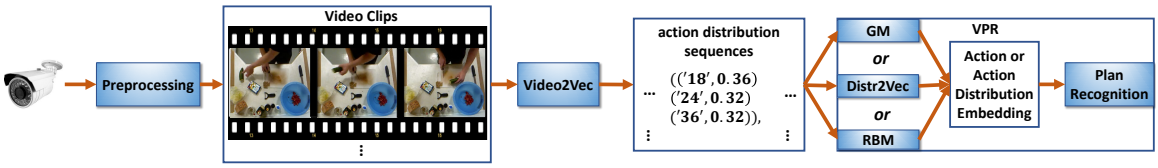


Figure 3.7: This Figure Illustrates the Framework of VPR, Which Does Plan Recognition on Uncertain Observations. Those Uncertain Observations Come From a Visual Recognition Model, Video2Vec, Whose Training Is Explained in the Supplemental Material (Section 1). The Number '18', '24', and '36' Means the 18-th, 24-th, and 36-th Action in Action Vocabulary (Explained in Section 3.3.1)

3.3 What if Learning with Erroneous Visual Recognition?

The previous work has an underlying assumption that there is a perfect perception module that provides optimal visual recognition results. But this assumption is too strict to be held in real-world settings. Visual activity recognition is often noisy and uncertain, typically represented as a probability distribution over possible actions. When there is an erroneous sensory, the action that matches the ground-truth would not be assigned with the highest probability. As a result, a high-level plan recognition module would not access such an action, but the one with the highest probability. However, if the plan recognition model uses the whole distribution, the plan recognition performance could be improved because the correct information is kept even at high-level computation.

Consider the example in Figure 3.7, where a surveillance system attempts to recognize plans from captured video clips of salad-making activities. In the first three frames, the activity recognition module may generate a distribution over three activities, and determine that the human grasps a pestle with the highest confidence (probability), a cucumber, and a pen with lower confidences. In such a case there is a perceptual error (PE), if we assume that the ground-truth activity is picking-up

a cucumber. If the plan recognition module only takes the most likely activity (i.e. grasping a pestle), only erroneous information is used by the plan recognition module which may lead to poor performance. Hence the plan recognition module should consider the information of less likely activities as well.

In our work [166], we tackle the problem of visual plan recognition. For most real-world problems, visual plan recognition is challenging in the sense of defining or learning a domain model for plan recognition. One approach to obtain such a model is to learn embeddings to capture affinities. Our framework uses an affinity model trained from the sequences of action distributions to do plan recognition. To use embeddings for plan recognition, we treat actions as words and learn an affinity model over actions based on their co-occurrences [145]. However, existing embedding methods expect sequences of words with no uncertainty (no distributions). One way to get such data is to greedily take the most possible action from each distribution, which motivates us to design a baseline model `GM` (Greedy Model). To use more information from uncertain observations, we can sample action sequences from each sequence of action distributions, and then build a `Word2Vec` model. Based on this we propose the Resampling-Based Model (`RS2Vec`). `RS2Vec` can be quite inefficient in that it will have to generate many samples to capture the distribution. The requisite sample size will increase with the length of the sequence, and the number of actions in the distribution of each step. An alternative is to directly handle the distribution data. We present such a method called `Distr2Vec`. `Distr2Vec` requires a modification of the `Word2Vec` architecture to let it learn embeddings for distributions over words as opposed to single words. To train the `Distr2Vec` affinity model, we introduce a loss function based on combining Kullback-Leibler (KL) divergence and Hierarchical-Softmax ([123]).

The resulting framework, called `VPR` (visual plan recognition), is described in our

paper. Figure 3.7 shows the architecture of VPR. A pre-trained activity recognition module converts a sequence of video clips from a camera, to a sequence of observed action distributions (i.e. observed plans). Then we could do plan recognition by using one of the three affinity models, GM, RS2Vec, or Distr2Vec. Those affinity models serve as approximated planning domain models and generate affinity scores for plan recognition. An affinity score represents how likely an activity is going to happen given observed activities.

3.3.1 Problem Setting

The problem formulation of recognizing plans from noisy and distribution-style action recognition is similar to the deterministic case explained below the Fig. 3.4. The main difference is that the input of our Distr2Vec in VPR is in the form of the following matrix. Each step is a distribution over actions $(a^1, ..., a^k)$ with their probabilities (confidence) $(c^1, ..., c^k)$. The values of T and K represent the number of time-steps and actions per step respectively.

$$\begin{array}{c} \\ \text{Actions} \end{array} \overset{\overset{\text{Time}}{\longrightarrow}}{\begin{array}{|c|}\hline \\ \\ \\ \\ \\ \\ \hline \end{array}} \begin{pmatrix} a_1^1, c_1^1 & a_2^1, c_2^1 & a_3^1, c_3^1 & \cdots & a_T^1, c_T^1 \\ a_1^2, c_1^2 & a_2^2, c_2^2 & a_3^2, c_3^2 & \cdots & a_T^2, c_T^2 \\ a_1^3, c_1^3 & a_2^3, c_2^3 & a_3^3, c_3^3 & \cdots & a_T^3, c_T^3 \\ \vdots & \vdots & \vdots & \ddots & \cdots \\ a_1^K, c_1^K & a_2^K, c_2^K & a_3^K, c_3^K & \cdots & a_T^K, c_T^K \end{pmatrix}$$

Fig. 3.8, 3.9, and 3.10 show an example for each of a plan library L, an observed plan O with missing observed action distribution, and a plan recognition solution R with predicated actions filled at each step with missing observations.

Plan1: $\begin{bmatrix} add\text{-}oil\text{-}prep & 0.7 \\ add\text{-}vinegar\text{-}prep & 0.2 \\ add\text{-}water\text{-}prep & 0.1 \end{bmatrix} \begin{bmatrix} add\text{-}oil\text{-}core & 0.7 \\ add\text{-}vinegar\text{-}prep & 0.2 \\ add\text{-}water\text{-}prep & 0.1 \end{bmatrix} \begin{bmatrix} add\text{-}vinegar\text{-}prep & 0.6 \\ add\text{-}oil\text{-}prep & 0.3 \\ add\text{-}water\text{-}prep & 0.1 \end{bmatrix} \begin{bmatrix} add\text{-}vinegar\text{-}core & 0.6 \\ add\text{-}oil\text{-}core & 0.3 \\ add\text{-}water\text{-}core & 0.1 \end{bmatrix} \begin{bmatrix} add\text{-}pepper\text{-}prep & 0.8 \\ add\text{-}cucumber\text{-}prep & 0.1 \\ add\text{-}salt\text{-}prep & 0.1 \end{bmatrix}$

Observation: $\begin{bmatrix} add\text{-}pepper\text{-}prep & 0.7 \\ add\text{-}cucumber\text{-}prep & 0.2 \\ add\text{-}salt\text{-}prep & 0.1 \end{bmatrix} \boldsymbol{\phi}\,\boldsymbol{\phi} \begin{bmatrix} add\text{-}vinegar\text{-}core & 0.6 \\ add\text{-}oil\text{-}core & 0.3 \\ add\text{-}water\text{-}core & 0.1 \end{bmatrix} \begin{bmatrix} add\text{-}salt\text{-}prep & 0.5 \\ add\text{-}vinegar\text{-}prep & 0.3 \\ add\text{-}oil\text{-}prep & 0.2 \end{bmatrix}$

Completed Plan: $\begin{bmatrix} add\text{-}pepper\text{-}prep & 0.7 \\ add\text{-}cucumber\text{-}prep & 0.2 \\ add\text{-}salt\text{-}prep & 0.1 \end{bmatrix} add\text{-}pepper\text{-}core\ add\text{-}vinegar\text{-}prep \begin{bmatrix} add\text{-}vinegar\text{-}core & 0.6 \\ add\text{-}oil\text{-}core & 0.3 \\ add\text{-}water\text{-}core & 0.1 \end{bmatrix} \begin{bmatrix} add\text{-}salt\text{-}prep & 0.5 \\ add\text{-}vinegar\text{-}prep & 0.3 \\ add\text{-}oil\text{-}prep & 0.2 \end{bmatrix}$

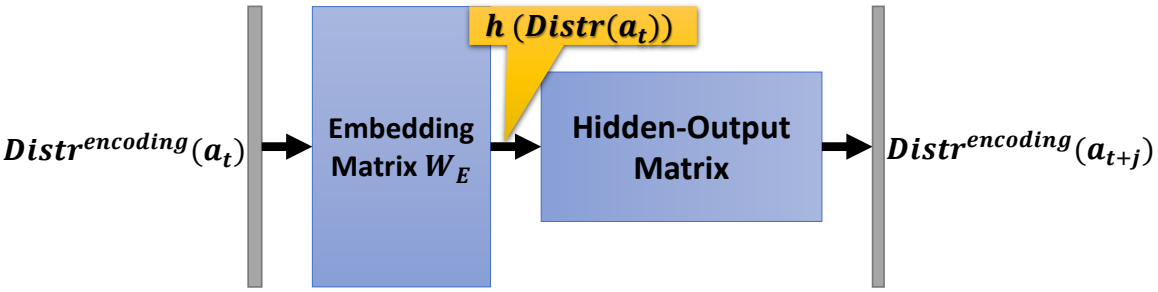


Figure 3.11: The Architecture of our Distr2Vec Model for Learning Distribution Embeddings and Action Affinity Models.

3.3.2 The Distr2Vec Model

Our `Distr2Vec` model framework is shown in Figure 3.11 which takes a distribution sequence like the one described above as input. Like the training of a Word2Vec model, we try to maximize the similarity (of the embeddings) between an action distribution and its neighbors. We minimize Kullback-Leibler (KL) divergence to maximize the similarity between our target and predicted output distribution as follows;

$$D_{KL}(Distr^{encoding}(a_{t+j})||\hat{Distr}^{encoding}(a_{t+j})) \tag{3.19}$$

where D_{KL} represents the KL divergence. KL-divergence is calculated as per Equ. 3.20.

$$KL(p||q) \triangleq \sum_{k=1}^{K} p_k log \frac{p_k}{q_k} = \sum_{k=1}^{K} p_k log p_k - \sum_{k=1}^{K} p_k log q_k \tag{3.20}$$

where q is the output probability distribution of our `Distr2Vec`. q is also an approximation of p, the target distribution $Distr(a_{t+j})$. We try to minimize the *inclusive* KL divergence ([87]) with the model's target distribution. An advantage of using inclusive KL divergence is that we avoid computing the derivative of the entropy of p when taking partial derivative of $KL(p||q)$ with respect to model parameters. This is because the values for p (which is $Distr(a_{t+j})$), is a constant with respect to the model parameters. Using this information, we can obtain the Equ. 3.21.

$$\begin{aligned} & D_{KL}(Distr^{encoding}(a_{t+j})||\hat{Distr}^{encoding}(a_{t+j})) \\ &= Z(Distr(a_{t+j})) - \sum_{k=1}^{K} c_{t+j}^k log p(a_{t+j}^k | h(Distr(a_t))) \end{aligned} \tag{3.21}$$

where $Z(Distr(a_{t+j})) = \sum_{k=1}^{K} c_{t+j}^k log(c_{t+j}^k)$ is a constant, and
$h(Distr(a_t))$ is the embedding computed by multiplying the embedding matrix W_E and the distribution input vector $Distr^{encoding}(a_t)$ (Equ. 3.22). $Distr^{encoding}(a_t) = \langle 0...0, c_t^1, 0, ..., 0, c_t^2, 0, ...0, c_t^K, 0... \rangle$ that is encoded from $Distr(a_t)$.

$$h(Distr(a_t)) = W_E \times Distr^{encoding}(a_t) \tag{3.22}$$

Now we elaborate how to combine Equ. 3.21 with using the hierarchical softmax ([123]). If we extend the hierarchical softmax in a normal Word2Vec to handle a distribution input $Distr(a_t)$, we obtain the probability of an action in the target observed action distribution $Distr(a_{t+j})$:

$$p(a^k_{t+j}|h(Distr(a_t))) = \prod_{i=1}^{L(a^k_{t+j})-1} \Big\{\sigma(\mathbb{I}(n(a^k_{t+j}, i+1) \\ = child(n(a^k_{t+j}, i))) \cdot v_{n(a^k_{t+j},i)} \cdot h(Distr(a_t)))\Big\} \tag{3.23}$$

And if we combine Equ. 3.23 and Equ. 3.21, we obtain the error function in Equ. 3.24. This is the error function as we are trying to minimize the KL divergence of Equ. 3.21.

$$E = Z(Distr(a_{t+j})) - \sum_{k=1}^{K} c^k_{t+j} \sum_{i=1}^{L(a^k_{t+j})-1} \\ \Big\{log\sigma(\mathbb{I}(.)v_{n(a^k_{t+j},i)} \cdot h(Distr(a_t)))\Big\} \tag{3.24}$$

Now with a trained `Distr2Vec` as the action affinity model, we use it as a subroutine in our `VPR`, which is similar to the usage of Word2Vec in `DUP` ([145]).

3.3.3 Resampling Based Model (RBM)

In `RS2Vec`, we first calculate the likelihoods of all possible paths from each trace of action distribution. The path weights (PW) can be calculated by multiplying the confidence values of all actions along a path, and thus could be used to represent the overall uncertainty of that path. The top N action sequences with probabilities (ranked according to PWs) are selected and then resampled using the roulette wheel resampling approach of [76]. This lets us drop some sampled traces that have low probabilities, and increase the number of samples with high probabilities. The set of traces selected after the resampling step is used to train a `Word2Vec` model. The above procedure is summarized in Algorithm. 3.

That said, the potential problem for `RS2Vec` are the following. First, the resampling would make the algorithm more computationally expensive, and the training

time would be closely related to the number of samples and length of the training data. This makes it hard to apply RBM to a real-time plan recognition system. Secondly, the resampling approach would lose some information, whereas `Distr2Vec` could directly use the entire distribution sequence.

3.4 Evaluation

3.4.1 Comparison between `RNNPlanner` *and* `DUP`

In this section we compare `RNNPlanner` with `DUP` to see the change of performance with respect to different distributions of missing actions in the underlying plans to be discovered. In this experiment, we are interested in evaluating the performance on consecutive missing actions in the underlying plans since these scenarios often exist in many applications such as surveillance [1]. We first test the performance of both `RNNPlanner` and `DUP` in discovering underlying plans with only consecutive missing actions in the "middle" of the plans, i.e., actions are not missing at the end or in the front, which indicates missing actions can be inferred from both previously and subsequently observed actions. Then we evaluate both `RNNPlanner` and `DUP` in discovering underlying plans with only consecutive missing actions at the end of the plans, which indicates missing actions can only be inferred from previously observed actions. After that, we also evaluate the performance of our `RNNPlanner` and `DUP` approaches with respect to the size of recommendation set. In the following subsections, we present the experimental results regarding those three aspects. The codes for evaluating `DUP` and `RNNPlanner` can be found at [1]

[1] `https://github.com/YantianZha/Discovering-Underlying-Plans-Based-on-Shallow-Models`

Algorithm 3 Algorithm for implementing RS2Vec model

INPUT: A library L of uncertain plans, and beam size S

OUTPUT: trained RS2Vec model that embeds action affinities in L

$SSP \leftarrow []$ ▷ Initialize stored sampled plans set SSP
for each uncertain plan p in L **do**
    $SS \leftarrow$heap_queue([]) ▷ Initialize stored samples set SS as a heap queue
    for $i = t; t \leq size(p); t++$ **do**
        $BSS \leftarrow$heap_queue([]) ▷ Initialize backup stored samples set BSS as a heap queue
        for each (a_t^k, c_t^k) from $Distr(a_t)$ **do**
            Copy each sample from SS, update its action sequence and score with a_t^k, c_t^k
            Push the updated sample into BSS
        end for
        if $S \leq BSS$ **then**
            Pop $size(BSS) - S$ samples from BSS
        end if
        $SS \leftarrow BSS$
    end for
    $SSP \leftarrow SSP \cup SS$
end for
Perform Rouletter wheel resampling on plans in SSP and keep the size of SSP to be unchanged
Train a Word2Vec on samples from SP and obtain `RS2Vec`
return `RS2Vec`

Performance with missing actions in the middle

To see the performance of `RNNPlanner` and `DUP` in cases when actions are missing in the middle of the underlying plan to be discovered, we vary the number of consecutive missing actions from 1 to 10, to see the change of accuracies of both `RNNPlanner` and `DUP`. We set the window size to be 1 and the recommendation size to be 10. The results are shown in Fig. 3.12.

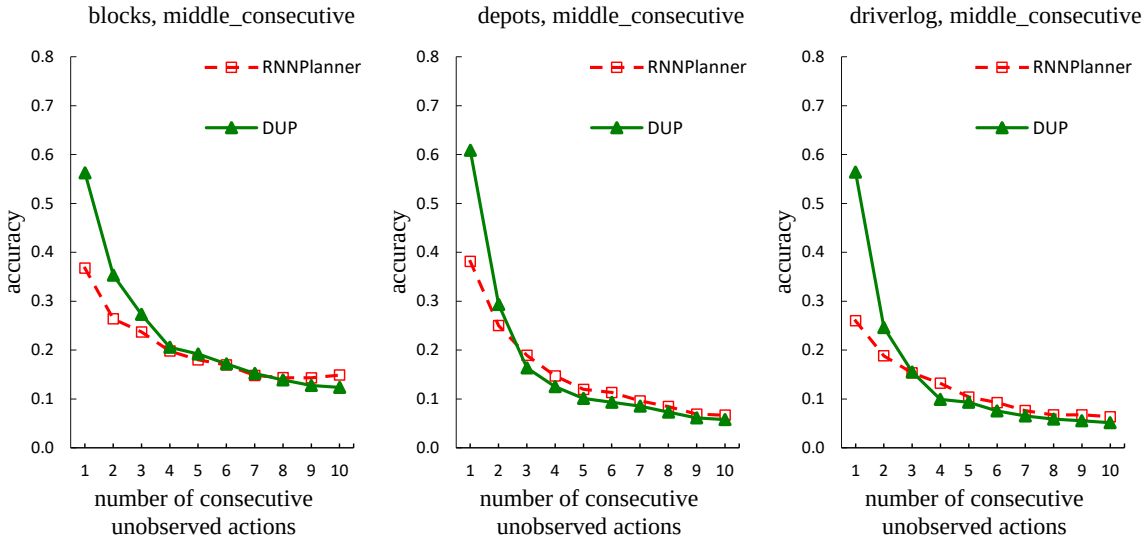


Figure 3.12: Accuracy With Respect to Missing Actions in the Middle

From Fig. 3.12, we can see that the accuracies of both `RNNPlanner` and `DUP` generally become lower when the number of consecutive unobserved actions increasing. This is consistent with our intuition since the more actions are missing, the less information can be used to help infer the unobserved actions, which results in low accuracies. Comparing the curves of `RNNPlanner` and `DUP`, we can see that the accuracy of `DUP` is higher than `RNNPlanner` at the beginning. This is because `DUP` exploits information of both observed actions before and after missing actions to infer the missing actions, while `RNNPlanner` just exploits observed actions before missing actions. When the number of missing actions is larger than 3, the accuracy of `DUP`

both low (i.e., lower than 0.2). This is because the window size of DUP is set to be 1, which indicates we exploit one action before the missing actions and one action after the missing actions to estimate the missing actions. When the consecutive missing actions are more than 1, there may not be sufficient context information for inferring the missing actions, resulting in low accuracy.

Performance with missing actions at the end

We also would like to see the performance of RNNPlanner and DUP in discovering missing actions at the end, which is prevalent in application domains that aim at discovering/predicting future actions. Similar to previous experiments, we vary the number of consecutive unobserved or missing actions to see the change of accuracies of both RNNPlanner and DUP. We set the window size to be 1 and the recommendation size to be 10 as well. The experimental results are shown in Fig. 3.13.

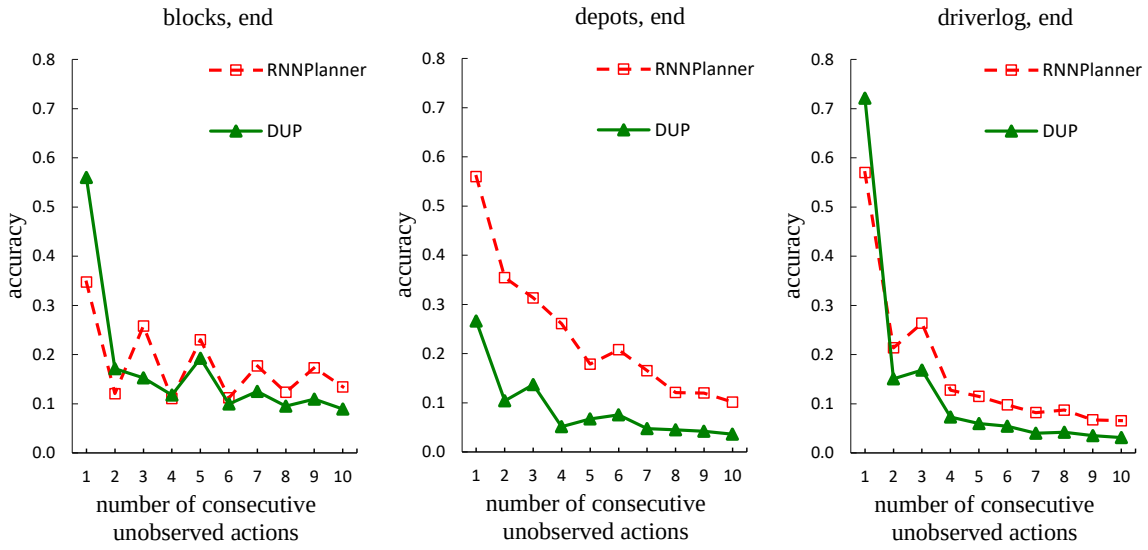


Figure 3.13: Accuracy With Respect to Missing Actions in the End.

From Fig. 3.13, we can observe that the accuracies of both RNNPlanner and DUP generally get decreasing when the number of consecutive missing actions increases.

This is similar to previous experimental results. That is, the more actions are missing, the less information is available for estimating the missing actions, which results in lower accuracy. In addition, we can also see that `RNNPlanner` generally performs better than `DUP`, which indicates that the RNNs-based approach, i.e., `RNNPlanner`, can indeed better exploit observed actions to predict future missing actions, since RNNs are capable of flexibly leveraging long or short-term information to help predict missing actions. We can also see that the performance of both `RNNPlanner` and `DUP` decreases sharply when the number of consecutive missing actions is larger than 3. This is because we set the window size to be 1 (i.e., we consider three consecutive actions each time we calculate the posterior probability), which indicates we exploit one action before the missing actions and one action after the missing actions to estimate the missing actions. When the consecutive missing actions are more than 1, there may not be sufficient context information for inferring the missing actions, resulting in low accuracies.

Performance with respect to different recommendation size

To see the change with respect to different recommendation size, we vary the size of recommendation sets from 1 to 10 and calculate their corresponding accuracies. We test our approaches with four cases: A. there are five actions missing at the end; B. there are five actions missing in the middle; C. there is one action missing at the end; D. there is one action missing in the middle. The results are shown in Fig.s 3.14-3.17 corresponding to cases A-D, respectively.

Case A: As shown in Fig. 3.14, `RNNPlanner` performs better than `DUP` mostly, except for when the recommendation set size is larger/equal to eight in blocks domain. This is because `RNNPlanner`, which contains LSTM cells, is able to actively

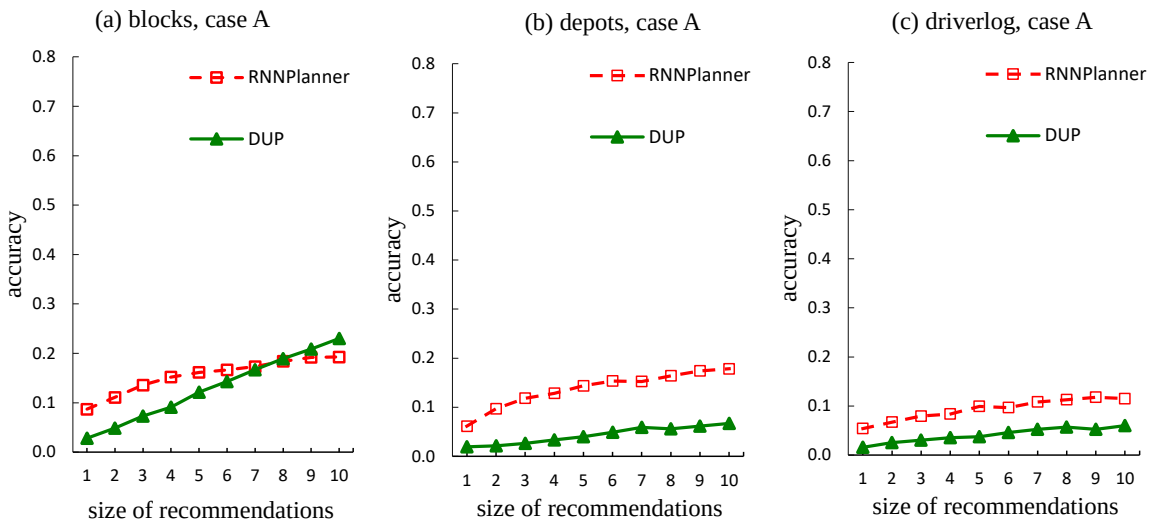


Figure 3.14: Case A: Accuracy With Respect to Different Size of Recommendations

remember or forget past observations (inputs) and computations (hidden states). For example, if in a set of sequences, a pattern A^{**} follows A^{*} after three words (i.e., $..., A^{*}, w_i, w_{i+1}, w_{i+2}, A^{**}, ...$), where w_i, w_{i+1}, w_{i+2} could be any word from the vocabulary except for A^{*} and A^{**}. And if the window size of DUP is smaller than three, then DUP is not able to utilize this pattern to predict A^{**} mainly based on A^{*}. When predicting the A^{**}, the DUP with context size one, works by searching for the most similar word to the w_{i+2}. One would yet argue that we can set a larger window size for DUP. Larger window size does not necessarily lead to higher accuracy, since using a larger window size also add more noise in the training of DUP. Remember that the word2vec model treats equally all possible word pair samples within its context window.

In addition, observing the accuracies (in terms of the size of recommendation S_x) of all three domains , we can see only in the blocks domain that DUP outperforms RNNPlanner, when S_x is larger than eight. Also in the *blocks* domain, DUP has the best performance, comparing to how DUP functions in other two domains.

This is because plans from the *blocks* domain has an overall higher ratio of #word to #vocabulary, which increases the possibility that the word pattern outside a context window, would reappear inside the window, and consequently help `DUP` recognize actions in the missing positions. Coming back to the example when we have a plan like $..., A^*, w_i, w_{i+1}, w_{i+2}, A^{**}, ...$, in *blocks* domain, it's more possible the word A^* happens again in one of w_i, w_{i+1}, and w_{i+2}.

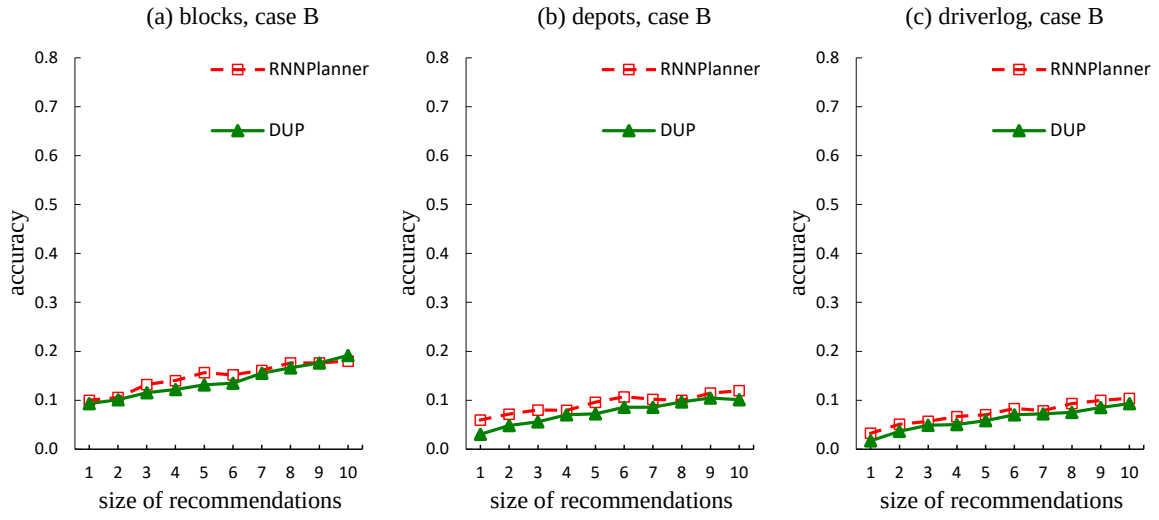


Figure 3.15: Case B: Accuracy With Respect to Different Size of Recommendations

Case B: What we can observe here from Fig. 3.15, is similar to our observations in ***case A***. `RNNPlanner` generally performs better than `DUP`, except for when the size of recommendation S_x is larger or equal to nine in *blocks* domain. It could also be observed that both `RNNPlanner` and `DUP` have the best accuracy performance in the *blocks* domain.

And by comparing the Fig. 3.15 in ***case B*** (five removed actions in the middle) with Fig. 3.14 in ***case A*** (five removed actions at the end), we can see that, the accuracy difference between `DUP` and `RNNPlanner` at each size of recommendation along the x-axis, is smaller in ***case B***. This is because, `RNNPlanner` only leverages

the observed actions before a missing position, whereas DUP has the advantage of additionally using the observation after a missing position.

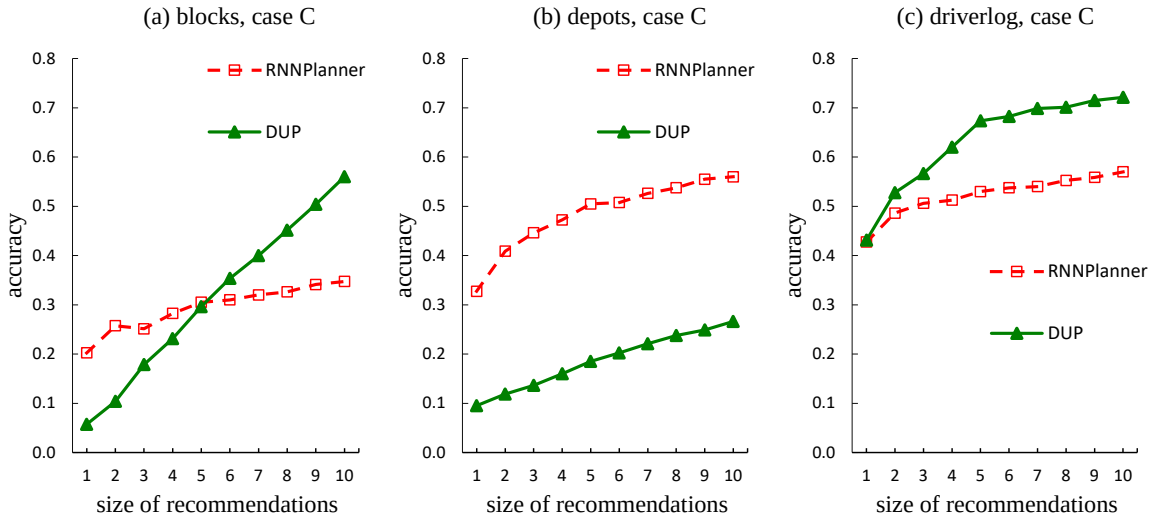


Figure 3.16: Case C: Accuracy With Respect to Different Size of Recommendations

Case C: From Fig. 3.16, we can see that both RNNPlanner and DUP could outperform each other in certain domains and recommendation set sizes (S_x). In *blocks* domain, DUP is better when S_x is larger than five. In *depots* domain, RNNPlanner is overwhelmingly better than DUP. In *driverlog* domain, DUP performs overall better except that, when there is only one recommendation, DUP is as good as RNNPlanner. This is because the domains *blocks* are *driverlog* are not as complex as *depots*, and DUP performs better in relative simple domains while worse in complex domains compared to RNNPlanner. In addition, if the number of consecutive unobserved action is less or equal to context window size (e.g., window size is one, and number of missing actions is one, in our ***case C***), then the fixed, and short context window of DUP, is competitive enough.

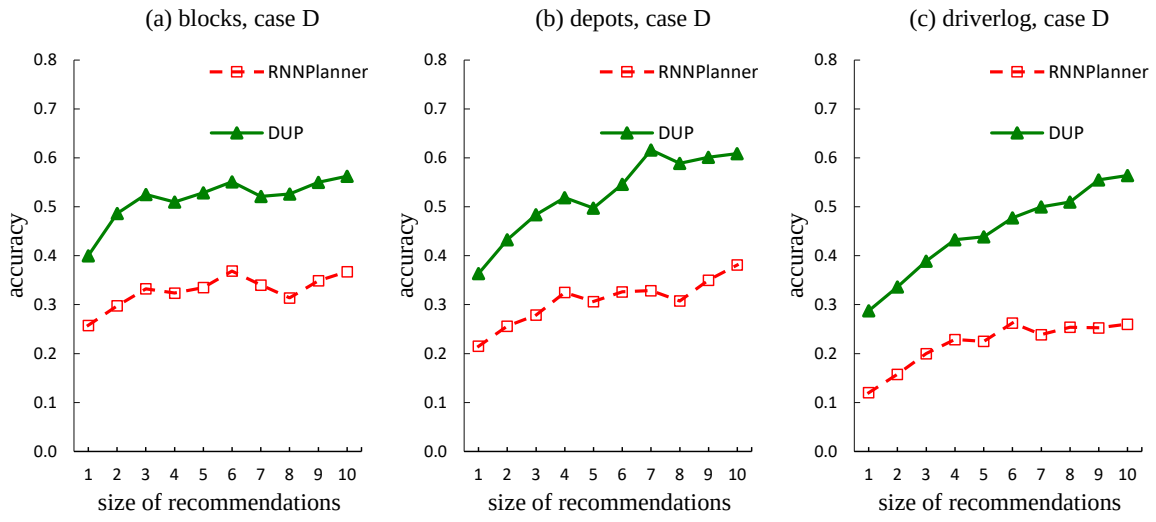


Figure 3.17: Case D: Accuracy With Respect to Different Size of Recommendations

Case D: From the results in Fig. 3.17, we can see that `DUP` functions better than `RNNPlanner` over all three domains, whereas `DUP` is worse in ***case A*** and ***case B***, and could occasionally be better than `RNNPlanner` in ***case C***. It makes sense in that, on the one hand, within the fixed and short context window, if there is very less positions with removed actions, `DUP` would have an improved performance. On the other hand, `RNNPlanner` is not able to leverage the information from both sides of a position with a missing action. Therefore, in ***case D***, `DUP` gains the benefit from both assumptions that there is only one missing action, and the position of that action is randomly chosen in the middle of a plan.

3.4.2 Evaluation of Distr2Vec

We evaluate the performance of `Distr2Vec` in `VPR` by comparing the performance with `VPR` that uses other affinity models trained with Word2Vec. To train a Word2Vec model from traces that have distributions, we sample from the distributions at each step and generate sampled traces of single actions (not distributions). The way we

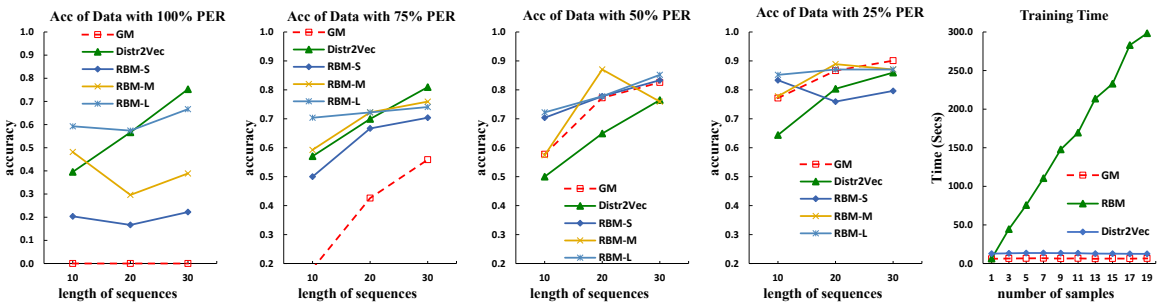


Figure 3.18: This Figure Demonstrates the Accuracy of GM, Distr2Vec, and Three RS2Vec Instances on Synthetic Data of Different PERs, With Respect to Increasing Plan (distribution Sequence) Lengths. The RBM-S, RBM-M, and RBM-L are RS2Vec Instances That are Trained on Small (S), Medium (M), and Large (L) Numbers of Samples.

sample the distributions gives us two baseline models: a Greedy Model (`GM`), and a Resampling Based Model (`RS2Vec`). In `GM`, we feed what the perception module considers the ground truth to Word2Vec for training affinity models. This is obtained by choosing the most probable action from each step's distribution. The codes for evaluating `Distr2Vec` in comparison with `RS2Vec` and `GM` in our `VPR` framework can be found at [2] .

In Figure 3.4.2, we show our plan recognition results for traces of length 10, 20, and 30, with varying PER and fixed entropy, in four sub-plots on the left. The RBM-S, RBM-M, and RBM-L are three `RS2Vec` instances that take three levels of samples (small, medium, and large) when being trained. We set the sample numbers of the small, medium, and large levels to 3, 9, and 27 respectively. In the rightmost sub-plot, we also show a comparison of **training time** required by `GM`, `Distr2Vec`, and `RS2Vec` with respect to an increasing number of samples that an `RS2Vec` instance takes when sequence length is 20. We did training time experiments for lengths 10, 20, and 30 of

[2] `https://github.com/YantianZha/Distr2Vec`

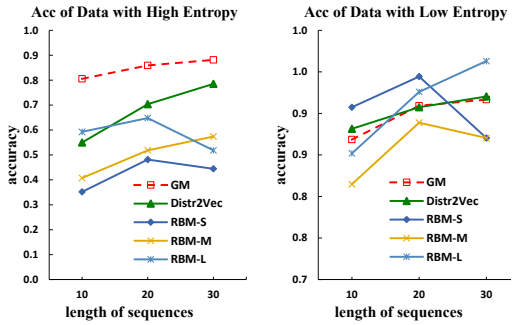


Figure 3.19: This Figure Demonstrates the Accuracy of GM, Distr2Vec, and Three RS2Vec Instances on Synthetic Data of Different Entropies, With Respect to Growing Sequence Lengths. The RBM-S, RBM-M, and RBM-L are Explained in the Description of Figure 3.4.2.

sequences, but because the three training time results are very similar to each other, we only report the length 20 results.

We observe that the training time of `RS2Vec` increases linearly with the number of samples per trace. This is easily explained by the fact that increasing the number of samples, is increasing the training costIn contrast, the training time for `Distr2Vec` and `GM` stay constant as there is no sampling step.

For plan recognition performances under all PER settings, `Distr2Vec` generally gets better performances for longer training sequences. As the trace length increases, `RS2Vec` requires more samples from the distribution sequence to match or outperform the `Distr2Vec` model (this is more obvious when PER is high). For `RS2Vec`, increasing the number of samples could be beneficial when PER is higher. Higher PER means more observed action distributions in a plan would have lower probabilities for correct actions (assuming each distribution must have one action that matches the ground-truth). Thus, taking more samples would give `RS2Vec` more access to correct information. We can observe clear patterns in sub-plots of 100% and 50% PER, that

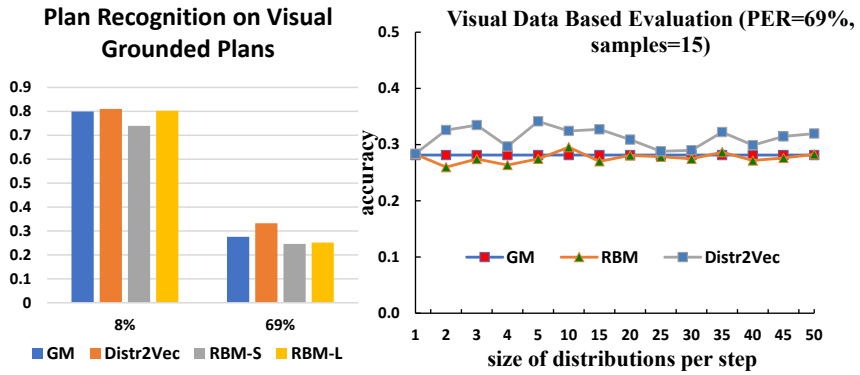


Figure 3.20: Results of Evaluating the GM, RS2Vec, and Distr2Vec on Videos, Which Demonstrate the Effectiveness of Distr2Vec. The Left Sub-plot Shows how Distr2Vec, GM, and RS2Vec With Small and Large Number of Samples Performs With Data of Different Entropies.

`RS2Vec` could get better performance by training on more samples.

As for the `GM` model, its accuracy for 100% PER across all lengths is 0%. This matches our expectations because the training data for the `GM` model does not have the ground-truth action in any step for 100% PER. Therefore, the `GM` would not capture any relevant information. For lower PER cases the `GM` performance is higher. This matches our expectations since the `GM` gets more of the correct information from the highest-probability sampling of low PER traces.

Lastly, for data with high and low entropies, the experimental results are shown in Figure. 3.19. We set PER to zero, and sample numbers of RBM-S, RBM-M, and RBM-L, to 7, 11, and 15 respectively. Thereby, the actions which match ground-truth ones are the most probable ones in all distributions. As expected, `GM` performs the best when the entropy of distributions in training distribution sequences (plan corpus) is high, since it gets the correct plan trace when PER is zero. The performance of `Distr2Vec` is expectedly lower than `GM`, yet it gets appreciably better than `RS2Vec`. In low entropy cases, all models perform comparably well.

We also evaluated the three models on the 50 Salads dataset which contains real-world videos of activity sequences. Figure 3.20 shows evaluation results on a plan corpus extracted from those salads-making videos. The left subplot shows how `Distr2Vec`, `GM`, and `RS2Vec` which takes 9 (as RBM-S) and 27 (as RBM-L) samples would perform, when PER is high (69%) and low (8%), meanwhile setting distribution size to 5. The right sub-plot shows how sizes of distributions per step influence the accuracy (setting PER to 69%). We observe that when PER is high, `VPR` with `Distr2Vec` clearly performs the best, whereas when PER is low, all models perform comparably well. This is consistent with the results of synthetic data evaluation. We also observe that overall the distribution size is not a key factor for deciding model performance.

3.5 Related Work

Our work is related to the four aspects on plan recognition, i.e., plan recognition with plan library, plan recognition with domain knowledge, senor-base activity/plan recognition, and planning with incomplete domain models. We will introduce the four aspects in the following subsections, respectively.

3.5.1 Plan Recognition with Plan Library

Our work is related to plan recognition with plan library. Kautz and Allen proposed an approach to recognizing plans based on parsing observed actions as sequences of subactions and essentially model this knowledge as a context-free rule in an "action grammar" [59]. All actions, plans are uniformly referred to as goals, and a recognizer's knowledge is represented by a set of first-order statements called event hierarchy encoded in first-order logic, which defines abstraction, decomposition and functional relationships between types of events. Lesh and Etzioni further presented methods in

scaling up activity recognition to scale up his work computationally [71]. They automatically constructed plan-library from domain primitives, which was different from [59] where the plan library was explicitly represented. In these approaches, the problem of combinatorial explosion of plan execution models impedes its application to real-world domains. Kabanza and Filion [56] proposed an anytime plan recognition algorithm to reduce the number of generated plan execution models based on weighted model counting. Avrahami-Zilberbrand and Kaminka [9] considered the observed agent to be an adversary and built an efficient hybrid adversarial plan recognition system that is composed of two processes, i.e., a plan recognizer capable of efficiently detecting anomalous behavior, and a utility-based plan recognizer incorporating the observer's own biases. Mirsky and Gal [88] proposed an efficient algorithm for online plan recognition called Semi-Lazy Inference Mechanism, which combined both a bottom-up and top-down parsing processes. Massardi et al. propose an anytime top-down approach [84] to deal with noisy observations based on the plan library. These approaches are, however, difficult to represent uncertainty and recognize plans that are not from the plan library. Bui et al. [18; 42] presented approaches to probabilistic plan recognition problems. Although we exploit a library of plans in our `DUP` and `RNNPlanner` approaches, we aim to learning shallow models and utilize the shallow models to recognize plans that are not necessarily in the plan library, which is different from previous approaches that assume the plans to be recognized are from the plan library.

3.5.2 Plan Recognition with Domain Knowledge

Instead of using a library of plans, Ramirez and Geffner [115] proposed an approach to solving the plan recognition problem using slightly modified planning algorithms, assuming the action models were given as input (note that action models

can be created by experts or learnt by previous systems [158; 175]). Due to the computational cost of calling a planner twice for each possible goal in [115], E-Martín et al. [33] proposed an approach that can quickly provide a probability distribution over the possible goals by computing cost estimates using a plan graph. Considering unreliability of a sensor malfunction or intentional obfuscation by malware, Sohrabi at el. [134] proposed a relaxation of the plan-recognition-as-planning formulation that allows unreliable observations. Pereira et. al. [106] proposed two goal recognition heuristics based on planning techniques that rely on planning landmarks [52]. Saria and Mahadevan presented a hierarchical multi-agent markov processes as a framework for hierarchical probabilistic plan recognition in cooperative multi-agent systems [128]. Singla and Mooney proposed an approach to abductive reasoning using a first-order probabilistic logic to recognize plans [132]. Amir and Gal addressed a plan recognition approach to recognizing student behaviors using virtual science laboratories [6]. Ramirez and Geffner exploited off-the-shelf classical planners to recognize probabilistic plans [118]. Albrecht and Stone recently provided a survey on plan recognition in three aspects, i.e., plan recognition in hierarchical plan libraries, plan recognition by planning in action models, and plan recognition by similarity to past plans [4]. To alliviate the requirement of complete domain models, Pereira et al. propose to recognize goals using incomplete domain theories by considering different notions of planning landmarks [107]. Different from those approaches, we do not require any domain model knowledge provided as input. Instead, we automatically learn shallow action models from previous plan cases for recognizing unknown plans that may not be identical to previous cases.

3.5.3 Sensor-based Activity Recognition

Activity recognition with sensors has been a major focus in the area of artificial intelligence. There has been an increasing interest in inferring a user's activities through low-level sensor modeling. Liao et al. applied a dynamic Bayesian network to estimate a person's locations and transportation modes [75] from logs of GPS data with relatively precise location information. Bui et al. introduced an abstract hidden Markov model to infer a person's goal from camera data in an indoor environment, but it is not clear from the article how action sequences are obtained from camera data [19]. Yin et al. explicitly relied on training a location-based sensor model [160] to infer locations from signals; the locations are part of the input that can serve as labels in the training data. To reduce the human labeling effort and cope with the changing signal profiles when the environment changes, Yin et al. dealt with the second issue by transferring the labelled knowledge [161] between time periods. Pan et al. proposed to perform location estimation [103; 104] through online co-localization, and apply multi-view learning for migrating the labelled data to a new time period.

Different from considering location information, Lester et al. propose to build user models [72] for different users and recognize user activities based on the models. They treat all of the users equally by simply mixing their data in training. However, different users may behave differently given similar sensor observations. For example, a user may visit the coffee shop for meal and the other just enjoys sitting in its outdoor couches to read research paper. These two users probably observe similar WiFi signals, but their activities are quite personalized. This implies that it may not be appropriate to require all of the users to share one common, user-independent activity recognizer. Therefore, Zheng et al. proposed to build a personalized activity recognition model [172] by considering the relations among users. Using sensor

data as input, Hodges and Pollack designed machine learning-based systems [51] for identifying individuals as they perform routine daily activities such as making coffee. Liao et al. proposed to infer user transportation modes [75] from readings of radio-frequency identifiers (RFID) and global positioning systems (GPS). Freedman et al. explore the application of natural language processing (NLP) techniques [37], i.e., Latent Dirichlet Allocation topic models, to human skeletal data of plan execution traces obtained from a RGB-D sensor. Bulling et al. discussed the key research challenges [20] that human activity recognition shared with general pattern recognition. When activity recognition is performed indoors and in cities using the widely available Wi-Fi signals, there is much noise and uncertainty. Xie et al. proposed a temporal-then-spatial recalibration scheme to build an end-to-end Memory Attention Networks (MANs) [153] for solving skeleton-based action recognition task. Chen et al. propose a multi-agent spatial-temporal attention model [23] to jointly recognize activities of multiple agents. To tackle the limitations of feature extraction and training data labeling effort, Qian et al. propose a distribution based semi-supervised learning approach [110] to recognize human activities.

Many different applications of activity recognition have been studied by researchers. For example, Pollack et al. show that home-based rehabilitation [109] can be provided for people suffering from traumatic brain injuries by automatically monitoring human activities. Chu et al. present a model of interactive activity recognition [25] to determine the user's state by interpreting sensor data and/or by explicitly querying the user. The system can be used in an asssistive system for persons with cognitive disabilities, which can prompt the user to begin, resume, or end tasks. Zheng et al. proposed to recognize physical activity from Accelerometer Data [173] Using a Multi-Scale Ensemble Method.

Different from the above-mentioned sensor-based activity/plan recognition, we do

not assume we have any labeled sensing data for recognizing plans. Instead, we focus on symbolic plan recognition in this thesis.

3.5.4 Sensor-based Intention Recognition

Another relevant body of research is intention recognition from visual observations [154; 61; 53]. The intention recognition modules in these works remove observation uncertainty by taking only the most likely observed (or demonstrated) trajectories and object states from a tracking module in a similar sense of our `GM`. They will therefore lose the information in the distribution-style observation sequences. Our work may be used to augment these and similar visual intention recognition works because our approach preserves the information from the uncertainty in the data.

3.5.5 Planning with Incomplete Action Models

Our work is also related to planning with incomplete action models (or model-lite planning [57; 174]). Fig. 3.1 shows the schematic view of incomplete models and their relationships in the spectrum of incompleteness. In a full model, we know exactly the dynamics of the model (i.e., state transitions). Approximate models are the closest to full models and their representations are similar except that there can be incomplete knowledge of action descriptions. To enable approximate planners to perform more (e.g., providing robust plans), planners are assumed to have access to additional knowledge circumscribing the incompleteness [148]. Partial models are one level further down the line in terms of the degree of incompleteness. While approximate models can encode incompleteness in the precondition/effect descriptions of the individual actions, partial models can completely abstract portions of a plan without providing details for them. In such cases, even though providing complete plans is infeasible, partial models can provide "planning guidance" for agents [170].

Shallow models are essentially just a step above having no planning model. They provide interesting contrasts to the standard precondition and effect based action models used in automated planning community. Our work in this chapter belongs to the class of shallow models. In developing shallow models, we are interested in planning technology that helps humans develop plans, even in the absence of any structured models or plan traces. In such cases, the best that we can hope for is to learn local structures of the planning model to provide planning support, similar to providing spell-check in writing. While some work in web-service composition (c.f. [32]) did focus on this type of planning support, they were hobbled by being limited to simple input/output type comparison. In contrast, we expect shallow models to be useful in "critiquing" the plans being generated by the humans (e.g. detecting that an action introduced by the human is not consistent with the model), and "explaining/justifying" the suggestions generated by humans.

3.6 Concluding Remarks

In this Chapter, we present two shallow-domain model based plan recognition approaches, `DUP` and `RNNPlanner`, based on vector representation of actions. For `DUP`, we first learn the vector representations of actions from plan libraries using the Skip-gram model which has been demonstrated to be effective. We then discover unobserved actions with the vector representations by repeatedly sampling actions and optimizing the probability of potential plans to be recognized. For `RNNPlanner`, we let the neural network itself to learn the word embedding, which would then be utilized by higher LSTM layers. We also empirically exhibit the effectiveness of our approaches.

One main weakness of the `DUP` and `RNNPlanner` as plan recognition approaches is that they assume that the visual recognition of activities must be accurate and

deterministic. If recognized actions are from a visual recognizer trained on real world data, the action recognition must have uncertainties. As such, there is a gap between plan recognition and real-world visual recognition. To make plan recognition more practically useful in scenarios with observational error in action recognition, we introduce our `VPR` framework that uses `Distr2Vec` and `RS2Vec` as action affinity models, to do visual plan recognition which may suffer from perceptual uncertainty. `Distr2Vec` and `RS2Vec` learn embeddings for distributions over actions to preserve the information from the visual recognition module. Our `VPR` framework can thus handle the uncertainty in visual data and perform plan recognition. We empirically show that when there is a higher Perception Error Rates (PER), `VPR` with `Distr2Vec` outperforms other models. When PER is lower, and sequence length is longer, all models perform comparably well. Another advantage of using `Distr2Vec` to handle distributions within the `VPR` framework is that the training time is shorter as compared to the `RS2Vec` model in which we have to sample more action sequences for comparable performance.

In my thesis, the works of learning shallow domain models for plan recognition works as a cornerstone for learning to couple perception and plan recognition. The `Distr2Vec` work presented in this Chapter bridges the gap between real-world visual recognition and plan recognition so that the plan recognition performance is improved. In the next Chapter, I investigate the question that can we use shallow-domain based plan recognition to improve perception?

3.6.1 Significance

The significance of my works in this Chapter lies at many broad directions to extend the shallow-domain based planning or plan recognition. In the future, it would be interesting to consider future studies as shown below:

1) While we focused on a one-shot recognition task in this chapter, in practice, human-in-the-loop planning will consist of multiple iterations, with `DUP`, `RNNPlanner`, and `VPR`, recognizing the plan and suggesting action addition alternatives; the human making a selection and revising the plan. The aim is to provide a form of flexible plan completion tool, akin to auto-completers for search engine queries. To do this efficiently, we need to make the `DUP`, `RNNPlanner`, and `VPR` recognition algorithms "incremental."

2) Another potential application for this type of **distributed or uncertain** action representations learned via `Word2Vec` or `Distr2Vec` is social media analysis. In particular, work such as [60] shows that identification of action-outcome relationships can significantly improve the analysis of social media threads. The challenge of course is that such action-outcome models have to be learned from raw and noisy social media text or video data containing mere fragments of plans or their executions. We believe that action vector models of the type we proposed in this chapter provide a promising way of handling this challenge.

3) In the presented works, we considered distributed representations of actions. In many real-world applications, however, more useful information could be in the form of images or texts describing "states" between actions is ubiquitous. It would be interesting to explore if we could leverage the representations of states, via deep learning models, e.g. [70], to help recognize plans together with adopting distributed representations of actions.

4) The plan to be recognized was assumed to be executed by a single agent in this chapter. We believe that the distributed representations of actions could be extended into handling team plan recognition [73].

5) Since the proposed `Distr2Vec` bridges the gap between visual recognition and plan recognition, `Distr2Vec` has a great potential of helping more complex cognitive

learning systems such as service robots that operate on visual sensory data. Such robots that typically cohabitate with humans need to make preactive decisions with considering humans in the loop.

Chapter 4

COUPLING BETWEEN PERCEPTION AND PLAN RECOGNITION

4.1 Interplay between Perception and Planning

Perception is about understanding the raw sensory data and convert them to more abstract and even semantic features that benefit higher-level information processing (e.g. planning or plan recognition). In this chapter, I only focus on visual sensory data. Visual perception has been one of the most popular research topics in Machine Learning and Computer Vision. The tasks of visual perception are diverse. Some of them aim at learning condenser and more expressive visual representations that can be accomplished by ConvNets, Auto-Encoders, etc. Some other works propose to learn high-quality mappings from raw video data to semantic abstractions, like symbols, concepts, activities, events, etc.

Traditional works that involve an integration of perception and planning usually treat planning and perception as two loosely-connected modules in a system. Once a perception model is trained, it will be fixed and treated as an external feature processing module that serves for high-level planning modules. Therefore, the perception-planning connections are typically uni-directional and fixed (engineered).

I would argue that the perception and planning modules could help overcome the limitations of each other. People tend to overlook that the perception module may also be improved by the planning or plan recognition module. One research topic that goes along this direction is active sensing, which leverages planning or sequential decision-making to motivate an agent to actively act to collect more valuable information and reduces its perceptual uncertainties. One way to achieve this is via updating

the estimation and observation in a Bayesian filtering framework, e.g. [43; 136], and the other way is by improving the quality of training data in a learning paradigm, e.g. [162]. However, here I would provide a different perspective: we can use plan recognition of agents to help generate a better attention map to support the visual recognition module.

4.1.1 Chapter Highlights

- In Sec. 4.2, I introduce the Pixel Dynamics Networks ([166]) that generates top-down attention maps driven by recognized human plans. Consequently, the performance of a visual perception task like event recognition could be improved.

4.2 Enhancing Perception via Attention Driven by Plan Recognition

In dynamic and multi-agent environments, being able to selectively focus on certain interesting visual regions is crucial for intelligent agents. An intelligent surveillance camera may monitor different people doing different activities in different areas. Such tasks require high visual loads. Selective attention ([142]) allows agents to avoid being overloaded, to make the best use of their intra-agent resources. The performance and efficiency are thereby expected to be improved. The attention driven by high-level and task-relevant information is called top-down attention (TDA). The generation of TDA maps generally involves more cognitive processes, consuming mental resources like memory, reasoning, and planning. Selective attention can be driven by top-down feedback, or simultaneously triggered by physical characteristics, using bottom-up attention (BUA). In the field of computer vision, researchers have traditionally delved into constructing combined BUA and TDA for images, using hand-crafted features ([94]). In recent years, a few works using neural networks to learn combined BUA and TDA models have also emerged, like [7; 45; 65]. Such models may work for static

images or video streams. **However, none of them considers if TDA could be driven by the recognized plans of other agents in the environment, which should be a natural consideration in multi-agent applications.** We name this kind of TDA as plan-recognition-driven attention (PRDA). In our work, we address the problem of generating PRDA maps from video frames and corresponding recognized plans.

PRDA may provide practically useful models of selective attention for agents that act in multi-agent environments. In such scenarios, interesting regions that are far away from each other are likely to be connected by a plan. The plan may be output from a plan recognition module, and thus could be a sequence of actions that an agent is expected to take shortly. Such plans contain useful long-term information for guiding the PRDA generation.One of the major challenges for using recognized plans is to collect clean observations of states and actions. This is traditionally addressed by human or object recognition and tracking. However, in highly cluttered and dynamic multi-agent environments, methods like object recognition and tracking would become unreliable. Humans and objects in the video could be too small to be reliably recognized. Human and objects in movements would even exacerbate the difficulty. Consequently, the collected observations could be incomplete and noisy.

This motivates us to propose a pixel dynamics network (`PDN`), which learns to directly predict the next state of an object at the position of a pixel. And the prediction is conditioned on both a pixel-observation and pixel-level motion. Therefore, it works like a dynamics model in control theory ([13]) – a model that can predict the next state of an object, if given the current state, and a motion applied to that object. Note that the attention PRDA is driven by the plans of agents, but the aforementioned pixel-level motions do not directly belong to a plan because there is no *intention* for object particles. Our `PDN` works by decomposing an action in a plan to

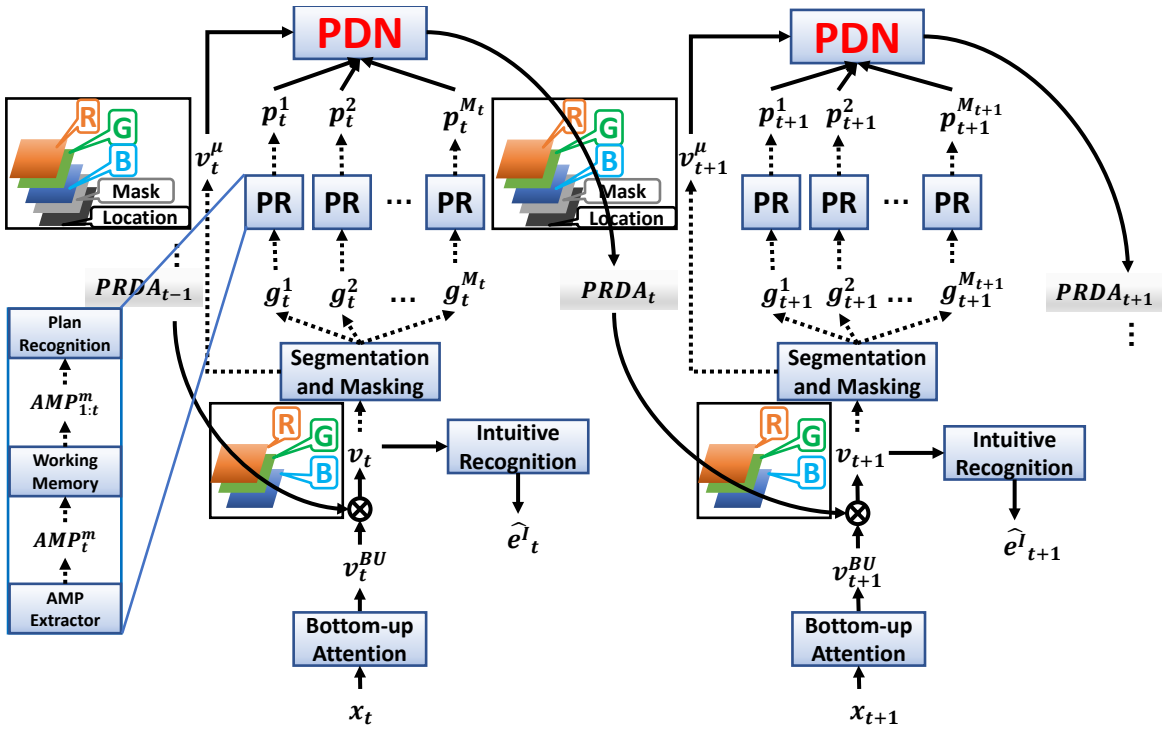


Figure 4.1: The Event Recognition System ER-PDN With Pixel Dynamics Network.

a group of pixel-level motions (we denote them as action conditional filters or ACFs). Then with a plan or a sequence of actions, we can convert each action to an ACF. Each ACF allows PDN to predicate a feature map as explained before. By performing channel-wise summarization through all pixel locations in those feature maps, we can obtain a PRDA map.

The usage of PDN is illustrated in Fig. 4.1. It starts from collecting action features from a plan. To do this, we extract frame-level motion features,from local video frames (video tubes) where an event happens. These video tubes could be obtained by temporally prolonging a bounding box. Then we do clustering on those features to learn a set of augmented motion primitives (AMPs). Each AMP vector represents an activity that happens in a video tube. After generating an ACF, `PDN` could take it with a local image patch to predict the next state of the object particle, which was

at the center location of that patch. We train the PDN in a weakly-supervised fashion by using an auxiliary task of event recognition. The overall training framework is named as ER-PDN which stands for **E**vent-**R**ecognition-PDN. As such, the learning of PDN has the access to reliable ground-truth training signals of the event label.

We evaluate ER-PDN on VIRAT Dataset ([102]) against a state-of-the-art visual recognition model [45].

4.3 Our Model

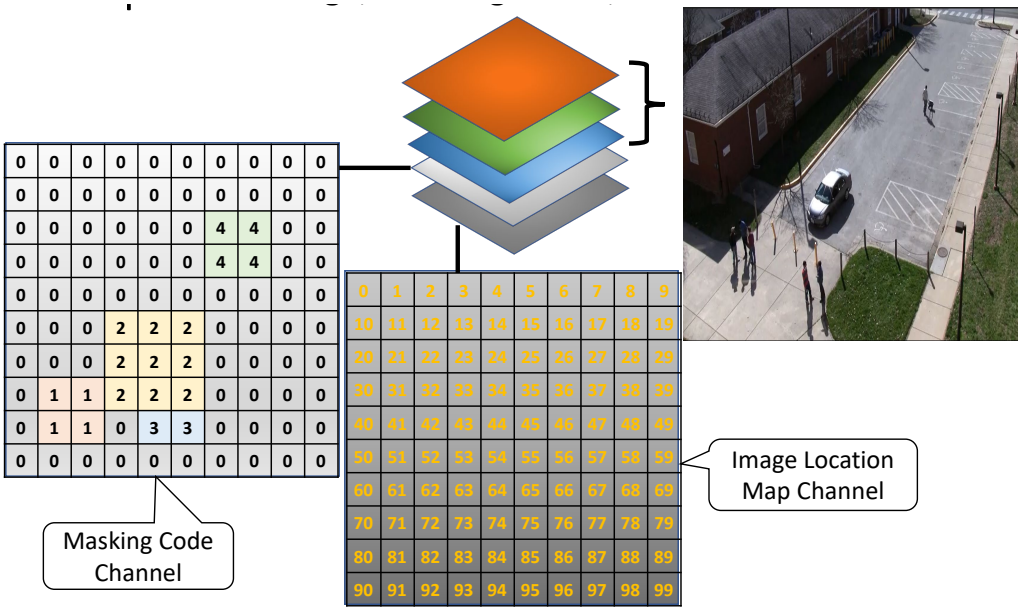
In this section we introduce our pixel dynamics network, and elaborate how it is used in event recognition.

4.3.1 Augmented Motion Primitives Generation

Now we introduce our way of generating AMPs as the representation of pixel-level actions. First, we extract features of HOG, HOF and HOD for each frame. Secondly, we do a K-Means clustering on the total features of the whole dataset. We set the number of clusters to A. We then assign the $\tilde{A}$ nearest centroids of every frame feature. Consequently, we convert a video sequence to an index distribution, which is based on the frame feature clustering. Each distribution has $\tilde{A}$ AMP indices. We also assign a probability value for each index to approximate the uncertainty of an activity. This probability is computed by using the distance between centroids and the corresponding feature. Lastly, we merge several consecutive and identical index distributions to one distribution. This merging of index distribution can make plan traces less redundant.

Thus, recognizing the plan of a group of agents would be realized by predicting a sequence of AMPs for a specific region. The region could either by generated by using bounding boxes, or using attention logits as in our work. Then we assign each

of these regions, an identical number m. This indicates that the m-th group agents are doing m-th events. If there are totally M such regions, we label each image pixel with those M numbers. This gives an extra channel (channel $c = 1$) of masking codes, which serves as a mapping from a recognized plan to the corresponding image area. We also add the second extra channel (channel $c = 0$) of pixel locations. After concatenating the two extra channels with original R, G, and B channels, we obtain a masked image, denoted as V^{μ}, as illustrated in Figure 4.2. Correspondingly, the Figure 4.3 illustrates the procedure of using a pair of local images (located by masking channel) to extract one AMP as in the red circle.



Plan Recognition

We adopted the work of Distr2Vec, [167], to generate future AMPs, given a sequence of observed AMP distributions. To elaborate, the plan recognition algorithm is based on a shallow planning model, Distr2Vec (distribution to vector), proposed in the work (Zha et al., 2018). Distr2Vec assumes observed action distribution sequences as

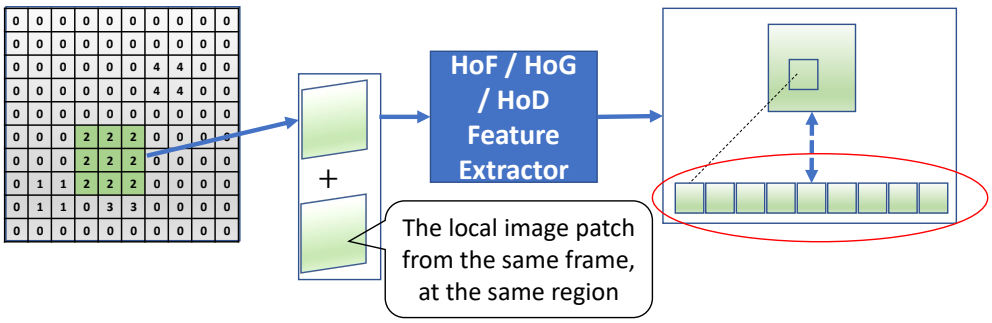


Figure 4.3: The Illustration of Extracting One AMP (Augmented Motion Primitive) From a Local Image Patch as a PDN Input.

inputs. This is a good fit for using noisy recognition outputs from a vision model , which consists of sequences of distributions over motion primitives. Distr2Vec then learns an action affinity model from the distribution sequences, by minimizing the distance between each of two action distributions. With this trained action affinity model, plan recognition (predicting future actions) is done, by searching for the most similar action (or motion primitive) to observed action distributions.

4.3.2 Pixel Dynamics Network

Our PDN also takes both a masked image, V^{μ} (Figure 4.2), and an AMP (Figure 4.3), as inputs. With these two inputs, the PDN would generate a state prediction map (SPM). The value at each location of the SPM represents the number of pixels that have been moved to that location by that AMP (action). This is like parallelly applying each of a group of dynamics models to each observation (a local image patch of pixels with corresponding locations) extracted from V^{μ}. The architecture of PDN is shown in Figure. 4.4. How PDN works could be divided into three stages: action conditional filter (ACF) generation (hidden output $h^{l=3}$ after third layer), convolution between ACFs and the input image to obtain state prediction values in a set of feature maps ($h^{l=4}$ after fourth layer), and a translation pooling applied to these feature maps,

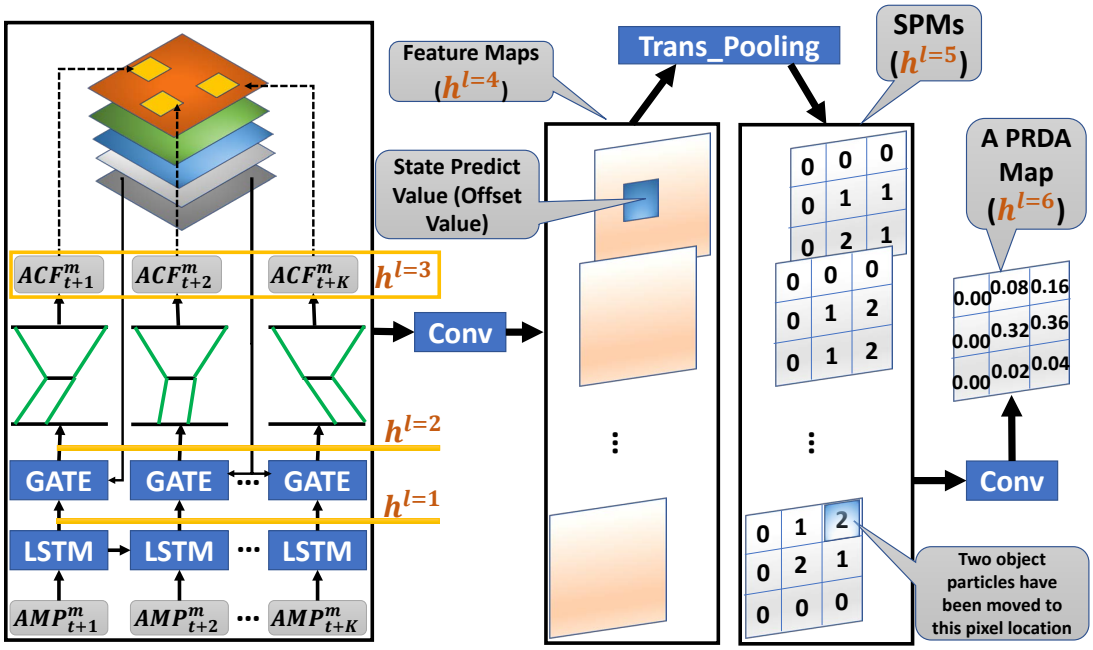


Figure 4.4: The Architecture of Pixel Dynamics Network.

to get a SPM ($h^{l=5}$ after fifth layer). We will introduce the three stages one by one in the following subsections.

Generation of Action Conditional Filter

Because we apply our PDN to generate attention maps driven by recognized plans, we feed the input action to a Long Short-term Memory (LSTM) layer. The LSTM layer could handle the sequential correlations among actions in a recognized plan. Note that an AMP contains pixel-level motion information in a local region (of one event). Hence, we cannot directly use the output of (the first) LSTM layer ($h^{l=1}$) as the filter to convolve with a local image patch. The local image patch being convolved might be smaller than a bounding box or a region with high attention logits. As a result, that local image patch only match a sub-region of the input AMP. This is why we have a gate layer. The gate layer is used to select a certain part of $h^{l=1}$ that matches

a local image patch, which leads to the need of inputting the channel $c = 0$, the pixel location map, to the gate layer. The output of (the second) gate layer, $h^{l=2}$, could be seen as a sparse version of the $h^{l=1}$. After applying the gate function, some parts of $h^{l=2}$ are not usable anymore. When doing convolution between $h^{l=2}$ and a local image patch, some parts in a local image patch would not contribute to the computation. The important information in a local image patch may be missed, which could be a problem. We address this problem by proposing a biased encoding-decoding layer that perform encoding-decoding on $h^{l=2}$, which reflects parts of $h^{l=1}$ selected by the gate layer. Thus, it does encoding (compressing) based on input part that is being paid attention to, and reconstruct with richer details (like zooming in). The reconstruction output per step is an ACF. Generating an ACF is to convert a meta action (AMP) to a group of pixel-level actions. $h^{l=3}$ consists of $M \times K$ ACFs. The equation for using the above procedure to compute an ACF, is shown in Equation. 4.1.

$$
\begin{aligned}
ACF^{m}_{k,t} = bias - decoding(bias - encoding(\\
LSTM(AMP^{m}_{1:k,t}) \odot \sigma(W^{GX} V^{\mu,c=0}_{t} + B^{G})), \\
1 \leq k \leq K, 1 \leq m \leq M
\end{aligned} \tag{4.1}
$$

where $ACF^{m}_{k,t}$ is the k-th action-conditioned filter generated from the k-th AMP in a plan of length K. That plan is the m-th recognized plan, of totally M plans. This is also the plan of m-th group of people that are doing m-th event. We further assume that all of the M recognized plans equally have length K. $LSTM(\cdot)$ denotes that the observed k AMPs are fed into a LSTM cell one by one, and output a LSTM state ($h^{l=1}$) per step. $V^{\mu,c=0}_{t}$ is the first channel of input image, containing a map of pixel locations. The values in $V^{\mu,c=0}_{t}$ would range from zero to the product of height of width of input image. The W^{GX} and B^{G} denote weights and bias parameters of the

gate layer. The $\odot$ between LSTM output and gate layer output denotes the element-wise multiplication. The result of this multiplication is fed into a bias-encoding and bias-decoding module.

Our bias-encoding and bias-decoding module works by performing a K-max pooling on the output from gate layer, which obtain a condensed vector, as shown in the small interval (above the gate module) between two green lines in Figure. 4.4. Then take that condensed input from K-max pooling as the hidden feature, and adopt normal auto-encoder framework to reconstruct the hidden vector to a feature, which is an ACF.

Generation of State Prediction Values

With $K \times M$ generated ACFs, the next task is to use each of them, to convolve with input image. Values in output feature maps after the convolution between an ACF and the input image, would be the offset values of the object particle at the center location of local image patch being convolved with an ACF. The convolution between an ACF and the input image could be described by the Equation. 4.2.

$$h_{ij}^{l=4} = (V_t^{\mu} \odot W^F * F_{k,t}^m)_{ij} = \sum_{a=0}^{F_1-1} \sum_{b=0}^{F_2-1} [[V_{i+a,j+b}^{\mu,c=0} = m]] \sum_{c=2}^{4} (ACF_{a,b}^{m,k} \cdot (V_{c,i+a,j+b}^{\mu} \cdot W_{i+a,j+b}^{l=4}) + B_{i,j}^{l=4}) \tag{4.2}$$

where $h_{ij}^{l=4}$ is the output from convolving ACFs with V^{μ}, i.e., the computation of fourth layer. $W_{i+a,j+b}^{l=4}$ and $B_{i,j}^{l=4}$ are weights and bias in the fourth layer, that is used for the convolution with value at location (i, j). F_1 and F_2 are respectively the height and width of each ACF. $ACF_{a,b}^{m,k}$ is the value at location (a, b) in the k-th ACF of m-th plan. As mentioned earlier, V_t^{μ} is an masked image at time t, with five channels:

pixel location map in channel 0, masking code map in channel 1, and RGB maps from channel 2 to 4. Particularly, the masking code map has values that match the index of a recognized plan. In a frame of image, if there are multiple groups of people doing multiple events, then each group matches a highlighted region, and a recognized plan. All pixels in that region are masked by the index of the recognized plan. The values of masking code are in the masking map. If there are M_t different recognized plans at time t (as shown in Figure. 4.1, there would be $M_t + 1$ different values in masking map. The reason is, we assign the value zero, to the regions of pixels do not belong to any of recognized plans.

Translation Pooling

Once obtaining a feature map of state prediction values, the next step is to convert that to a SPM, which is accomplished by our translation pooling layer. A state prediction value at a position represents the offset as explained before, and the translation pooling layer works by converting that offset value, to a "plus one" operation at a new position, after applying the offset value to the old position. The algorithm of translation pooling is shown in Algorithm. 4.

4.3.3 PRDA Generation Layer

The role of this layer is to pixel-wise summarization across all channels of SPMs (feature maps) from the translation pooling layer. This summarization will be taken as a PRDA map. Intuitively, as each of those SPMs maps to an action of an agent, summarization of SPMs would provide useful information of which visual region is likely to get considerable amount of pixel objects. As mentioned before, a pixel object means the object point at a location with the corresponding pixel as the observation. The PRDA generation layer also needs to output a feature map that has the same

Algorithm 4 Translation Pooling

procedure TRANS-POOLING($h^{l=4}, shape(N, N)$) ▷ From a set of $N \times N$ feature maps, in which values are computed using Equ. 4.2

    $h^{l=5}_{0:N\times N-1} \leftarrow 0$ ▷ Initialize all values in the vector $h^{l=5}$ (size is N^2) to zeros

    for $i = 1; i \leq N; i++$ **do**

        for $j = 1; j \leq N; j++$ **do**

            $h^{l=5}_{ID(i,j)-h^{l=4}_{ij}} += 1$ ▷ Generate statistics using all state prediction values in $h^{l=4}$

        end for

    end for

    $SPM \leftarrow reshape\ h^{l=5} : from\ (N^2)\ to\ (N, N)$

    return SPM ▷ Return a state prediction map

end procedure

procedure ID(i, j) ▷ Convert 2D index to 1D index

    return $i * N + j$

end procedure

size of a SPM and V^{μ}. We design a special ConvNet layer with filter size, weight, and slide stride set to one. And the weights in this layer are not trainable. This way, we are able to do summarization over all pixel regions, and across all SPMs, which is identical to Equation 4.3.

$$h^{l=6}_{ij} = PRDA^{t+1}_{i,j} = \frac{\sum_{c=0}^{M\times K-1} SPM^{t,i,j}_{c}}{Z} \tag{4.3}$$

where Z is a normalization number, and we set it to one in our experiments. In $h_{l=5}$, there are $M \times K$ SPMs, because there are M plans of length K, and each AMP leads to a SPM. Each SPM has $N \times N$ values. Generating a PRDA map requires

calculating statistics over all of them. In addition, here c denotes the c-th channel or SPM in $h_{l=5}$, t denotes the current time step, and each pixel location in a PRDA map is identified by i and j.

4.3.4 Event Recognition with PDN

Here we show how our PDN can be used to generate PRDA maps to support an event recognition task. We denote our event recognition system as ER-PDN. The overall framework is shown in Figure. 4.1. The solid paths are differentiable, and dashed ones are not. At each step, a video frame X_t, is filtered by bottom-up attention model, to get a BUA map V_t^{BU}. The V_t^{BU} is then combined with a PRDA map from the previous step, to get a highlighted frame V_t. The segmentation and masking component then extracts M_t glimpses, which are disconnected highlighted areas in V_t. Each of these glimpses are fed into a separate but structurally identical plan recognition component, PR, to generate M_t recognized plans p. The segmentation and masking component also adds additional channels as explained before, to map each of M_t recognized plans that is indexed by m, to a highlighted region in V_t^{μ}. Consequently, every pixel in V_t^{μ} also has a mask code m. The PDN takes all recognized plans for different highlighted areas, and the masked V_t^{μ}, to generate the PRDA map $PRDA_t$.

The above procedure describes the non-differentiable path (the dashed ones). In order to train the model, we want to let the model to predict a label so that there is a teaching signal. This is what the dashed path is showing. After generating V_t^{μ}, an intuitive recognition module is required to predict an event label at each step. The intuitive recognition module is a LSTM layer, which has the advantage of handling sequential inputs with internal long-term dependency.

4.4 Evaluation

In this section, we provide experiment results and analysis. We compare our PDN with a baseline vision model [45], by using them to solve an event recognition task. Note that the baseline model also has an attention layer. Unlike existing event recognition works which only use event-relevant sequences [150; 82], we did not do pre-segmentation or use bounding boxes. We ran our experiments on a machine with a Quad-Core CPU (Intel Xeon 3.4GHz), a 64GB RAM, a GeForce GTX 1080 GPU, and Ubuntu 16.04 OS.

4.4.1 Dataset Pre-processing and Training Procedure

We used the VIRAT 1.0 Ground Dataset [102] for evaluation. We randomly select 70% of all videos for training, and the rest for testing. And we down-sampled videos to use every six frames. We used different pre-processing procedures for learning augmented motion primitives (AMPs), and learning the PDN.

To learn motion primitive clusters, we applied K-means clustering to find the motion primitive cluster library. The motion primitive cluster library is a library of mappings between a motion primitive vector, and a cluster index. We set the number of clusters to 512, and apply clustering on HoD features collected from 70% of VIRAT videos for training. We did not use the original VIRAT videos. Instead, we used the video segments that are inside ground-truth bounding boxes. This means that the videos we use are both temporally and spatially smaller than the original videos. Those bounding boxes highlight event relevant frames and objects. In contrast, when training the ER-PDN, we did not use any bounding box information in videos, which significantly increases the event recognition difficulty. Once we obtain the AMP clusters, we can convert sequences of HoD features to a sequence of motion primitive

(or AMP) indices. Each of such a motion primitive index sequences could be seen as a high-level plan. Then we use the work of [167] to learn a shallow plan recognition model. When training the plan recognition model, we set the distribution size to three, and training epoches to sixty.

When training ER-PDN and the baseline model [45], We feed both models with resized images (450×450). Note that if there are multiple events simultaneously happening, then some video frames may map to multiple labels. For this situation, we copy certain images multiple times, and assign each with a corresponding event label. The video down-sampling allows us to increase the variance between each of two frames in our pre-processed dataset, and thereby increase the variance between each of two AMPs in a recognized plan.

When we train the ER-PDN, we feed four frames as a batch to the pre-trained plan recognition module to generate a K-step recognized plan (we set K to five). With two consecutive frames, the HOD feature extraction component generates a HoD feature vector, which would be used for searching top-3 AMP indices from the trained AMP cluster library. Thus, we could obtain a sequence of three AMP index distributions (each distribution has three AMPs), given an input image batch. This is similar to the work [167]. This AMP distribution sequence is used as an observed plan, fed into the plan recognition module, which returns a k-step sequence of future AMP indices that is most likely to happen (the recognized plan). ER-PDN then converts the future AMP indices back to a sequence of motion primitive feature vectors, by using the AMP cluster libraray. ER-PDN then feeds these vectors one by one, to generate proper ACFs, and SPMs. This is how ER-PDN uses input actions (AMP vectors).

To generate a PRDA, ER-PDN also takes an image as input. After taking a batch of four consecutive images from a video, ER-PDN uses the first image to generate

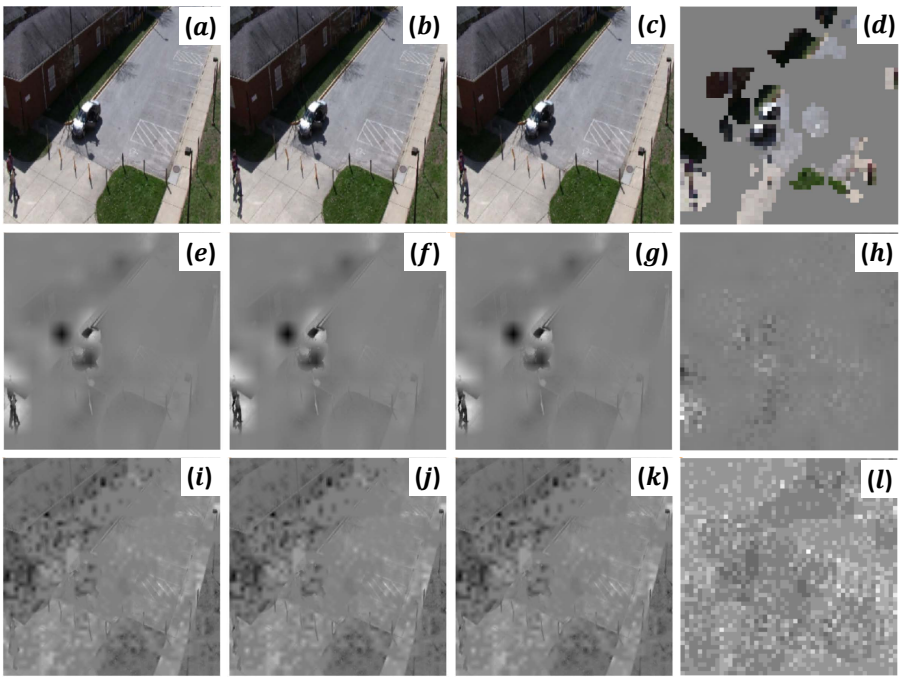


Figure 4.5: Results Visualization of our Event Recognition System With the PDN.

a BUA, which is then used to generate glimpse for plan recognition. And ER-PDN uses the last image, with the generated PRDA and BUA map, to augment the input and predict which event is happening.

4.4.2 Results and Analysis of Event Recognition

Model	Mean Average Precision
Baseline [45]	29.06%
ER-PDN	29.15%

Table 4.1: The Statistics of Event Recognition Performance on VIRAT Dataset, Using the Baseline Model [45], and our ER-PDN.

A comparison of the results in Table 4.1 shows that the visual recognition could be improved by a plan (a sequence of motion primitives). In addition, Figure 4.5 contains

examples of input video frames, and the corresponding feature maps of BUA, PRDA, and V_t^{μ}. Specifically, in Figure 4.5, (a), (b) and (c) are resized video frames. (d) is a visualization of V_t^{μ}, by applying a BUA to a resized frame and keeping parts within a threshold. (e), (f), and (g) are visualizations of applying a BUA to a resized frame. (h) is the feature map of applying both a BUA and a PRDA to the output of a convolutional layer, with an input frame. (i), (j) and (k) are visualizations of applying a PRDA to input video frames. (l) is a PRDA feature map.

We can observe from (a), (b) and (c) in Figure 4.5 that, three people are moving toward a car. (e), (f), and (g) in Figure 4.5 indicate that, the BUA map focus properly to interesting image areas, where there is a car, and people. Because the car is not moving, in (i), (j) and (k), the area with car is darker, which indicates less attention. The area between the three people, and the car, is lighter, which suggests that the moving people may want to get to the car. Thus, the lighter region between the people and the car indicates that there is more attention put there.

4.5 Related Work

4.5.1 Soft Attention Driven by High-Level Signals

Here, we briefly introduce previous work on using neural networks to generate soft attention maps, driven by high-level (or top-down) signals. The work [45] proposed an attention pooling layer to replace the normal pooling layer in ConvNets. Their high-level signal comes from class labels. For some other previous works that combine top-down and bottom-up attention, their top-down attention model only imitates how spatial reasoning influences attention. In [94], they use target object locations and surrounding distractor feature in an image as high-level signals to guide the top-down attention for visual search tasks. In [65], they use the appearance feature of targets.

In contrast, our planning-driven attention model exploits both spatial and temporal reasoning for long-term future, based on the awareness of environment dynamics and recognized plan feature of other agents.

4.5.2 Pixel-wise Classification

The design of PDN is also inspired by pixel-wise classification problems (i.e., semantics segmentation). Semantics segmentation is the vision problem that requires assigning a proper semantic label for each pixel. This could be accomplished by using ConvNets [77; 100; 22; 24; 91]. Our PDN, instead, assigns a "state offset" value for each pixel.

4.5.3 Plan Recognition

One of the main novelties of our work is to use high-level signals (recognized plans) to form better attention maps. Plan recognition is about predicting the future action sequences of an agent, given its observed actions. While traditionally people rely on classic planning approaches with a domain model to do plan recognition [116], recent years have seen an increasing interest in doing plan recognition on a learned and approximated model. Zhuo et al., [146] treat all plans in a plan library as sentences. Then they use the plan library as the training corpora to train a Word2Vec model. The plan recognition task then becomes to find the action (in the action vocabulary) that is most similar to surrounding actions in a testing plan. The similarity value is provided by the learned Word2Vec. Most recently, [167] extends the problem range of aforementioned work to using action distribution sequences as plans to learn a Distr2Vec model to handle observational uncertainties.

4.6 Concluding Remarks

Intuitively, we improve perception to improve high-level planning or plan recognition. In this chapter, I present a work that uses plan recognition to improve visual recognition (perception) tasks, via modeling Plan-Recognition-Driven-Attention. Essentially, the recognized plans contain long-term information that a perception agent could use to focus on proper regions that are closely relevant to a task.

This Chapter couples lower-level cognition functions like perception and conscious-level cognition functions like planning together, which makes the perception task-driven. In the next Chapter, I will present if we could use even meta-level cognitive modeling (e.g. introspecting and self-explaining) to improve the learning of lower-level cognition functions like perception and control.

4.6.1 Significance

One interesting insight for the Plan-Recogntion-Driven-Attention work is that it essentially motivates agents to **think ahead to better see the present** – it predicts future actions to guide the perception. This opens rich opportunities for future works:

1) While in this chapter, we use event recognition as a testbed perception task, there are other interesting tasks as well. For example, one can consider if the affordance prediction mentioned in Chapter 2 can be task-relevant – that is, conditioned on a planning solution of future activities for a task. If the planning solution is plan recognition results of human collaborators, then the resulting affordance prediction could help robots make decisions that better collaborate with humans.

2) In this chapter, we consider a pretrained shallow-domain based planner that provides action distribution representations for the perception agent. Future works could consider jointly learning action representations and visual tasks so that the

action representations could better fit perception agents. Note that a planner that can solve planning or plan recognition tasks is still needed for long-term reasoning.

Chapter 5

SELF-EXPLANATION GUIDED LEARNING

5.1 Self-Explaning: Figuring out "How" and "Why" for "Self"

Making deep learning models more interpretable has been a very popular research direction since five years ago. Deep neural networks usually work as if a black box that output a prediction from an input. However, interpretable deep neural networks offer explanations regarding how and why it gives a prediction as well. Such explanations help a human user or programmer understand the underlying computation process of the neural networks better. This in turn enables a human to improve the design of deep neural networks. Now a natural question to ask is, can machine-generated explanations be directly used by an agent to improve itself?

If an agent generates explanations based on its current knowledge for improving the functioning/learning of itself, we call such explanations as self-explanations, which lie in the intersection between Interpretable Machine Learning and Cognitive Psychology. As introduced in Sec. 1.2.3, self-explanation is essentially a kind of metacognition that monitors and improves the overall cognition (e.g. perception and performance) of an agent. The Cognitive Psychology community also researches the benefits of doing self-explanation for humans. As discovered by [131] (p. 53-54), such benefits can be summarized as follows: 1) It encourages human learners to explain and helps learners by increasing the depth of their explanations; 2) By self-explaining, learners could more often generate conceptually more sophisticated solutions; 3) Evidences also shows that self-explaining increases the range of strategies that a learner attempt; 4) The process of doing self-explanation involves changing the accessibility

of effective and ineffective ways of thinking; and 5) Self-explaining raises learners' degree of engagement with the task.

Inspired by the above benefits, we can conjecture that integrating self-explanation into the learning of neural networks could have similar benefits. First, self-explaining helps a neural network learner to discover the underlying task logics, which can guide the learning. This would probably be useful and even necessary when the environment and task could easily lead to ambiguous experiences. This matches the benefit 4) in the previous paragraph. The second benefit is that self-explanation inspires the neural network learner to explore in a more efficient fashion, similar to the benefit 3) in the previous paragraph. Third, by doing self-explanation, the learning could be regularized by preventing the agent from taking actions with a lower interpretability score, at each learning step. This matches the benefit 2) in the previous paragraph. All of the aforementioned benefits could origin from a reduced hypothesis space due to performing self-explanation with some prior knowledge.

A formulation of improving agents' learning with self-explaining is described by Equ. 5.1, 5.2 and 5.3.

$$domain\_knowledge \wedge inputs \wedge targets \models \mathbf{Train}(\mathbb{SE}) \tag{5.1}$$

$$domain\_knowledge \wedge \mathbf{Train}(\mathbb{SE}) \wedge inputs \models \mathbb{SE} \tag{5.2}$$

$$\mathbb{SE} \wedge inputs \models targets \tag{5.3}$$

where $\mathbb{SE}$ denotes a Self-Explaining function, $\models$ denotes an "imply" relationship, $targets$ can be seen as a final solution of the current task, **Train** denotes a training algorithm or paradigm.

The Equ. 5.1 expresses that the searching of a good training algorithm for a self-explainer $\mathbb{SE}$, which could be done by human experts, requires $domain\_knowledge$ and some examples of inputs and target. Such a training algorithm $\mathbf{Train}(\mathbb{SE})$ can

be viewed as a *solution_knowledge* that could train a self-explainer in a way that best matches the current task. Humans may have an ample resource of solution knowledge, but some examples of inputs and targets could help humans find the *solution_knowledge* that matches the current task. The *domain_knowledge* is task-agnostic and describes objects and their relationships in the world. In Equ. 5.2, the *domain_knowledge* may help the learning/construction of a Self-Explainer ($\mathbb{SE}$) by providing a smaller hypothesis space. **Train**($\mathbb{SE}$) would still need to be used to learn how to map a new input to somewhere in such a reduced hypothesis space.

5.1.1 Chapter Highlights

- In Sec. 5.2, I will discuss a work that uses self-explanation to help robot learn from ambiguous demonstrations – **S**elf-**E**xplaination for **R**einforcement-**L**earning **f**rom **D**emonstration (**SERLfD** [165]).

5.1.2 Background

Reinforcement Learning and Reward Shaping

In our work we consider a finite-horizon and discounted Markov decision process (MDP) model that can be learned by RL methods. In an MDP, the agent observes the current state $s_t \in S$ at step t, and then takes an action (or makes a decision) $a_t \in A$. The action would influence the environment and lead to a reward $r(s_t, a_t)$, a new state s_{t+1}, and a binary value "done". When "done" is True, the current episode ends. The above procedure is an interaction and generates an experience, defined by a 5-tuple $(s_t, a_t, r_t, s_{t+1}, done)$. Since the MDP is finite-horizon, each episode allows at most T steps of interactions. The agent is trained with the objective of finding a parameterized policy ($a \sim \pi_\theta(s|\theta)$) that maximizes the total accumulative rewards (Equ. 5.4).

$$\max_{\theta} \mathop{\mathbb{E}}_{s,a\sim\pi_\theta}[\sum_{t=1}^{T}\gamma^{t-1}r(s_t,a_t)] \tag{5.4}$$

where $\mathbb{E}$ denotes expectation, γ denotes the reward discounting factor.

In our work, we further assume that the reward function $r(s_t, a_t)$ is sparse, i.e., $r(s_t, a_t) = 0$ in most of the states $s \in S$. Training RL agents in sparse rewards environments could be challenging due to the delayed training signals from effective feedback. One extensively adopted way to ameliorate such training is by adding reward shaping, which provides denser training signals so that the agent could obtain valuable feedback much sooner. However, we need to be careful at providing shaped rewards. As demonstrated in [98], a poorly-designed reward shaping function may cause the converged optimal policy to shift as against the one under original rewards. [98] proves that potential-based reward shaping function, which follows the form in Equ. 5.5, is the only class of reward shaping function that can guarantee the invariance of optimal policies.

$$r'(s_t, a_t) = r(s_t, a_t) + \lambda\Phi(s_{t+1}) - \Phi(s_t) \tag{5.5}$$

where $r'(s_t, a_t)$ and $r(s_t, a_t)$ denote the shaped and original reward respectively, λ is an adjustment parameter, Φ denotes any real-valued function, and $\lambda\Phi(s_{t+1}) - \Phi(s_t)$ is the reward shaping term. Note that **we set λ to 1 in the rest of paper**.

Inverse Reinforcement Learning (IRL)

IRL is about the problem that the reward function in an MDP model is unknown and needs to be discovered from expert demonstrations. Typically in an IRL framework, the unknown reward function is modeled by a parameterized function of certain features. The design of our SERLfD framework is based on the Guided Cost Learning

(GCL [36]) optimized by Generative-Adversarial Networks (GAN-GCL [35]). GCL and GAN-GCL propose to couple maximum entropy IRL [177] and RL together to do continuous control learning with unknown rewards. In the maximum entropy IRL [177], a reward function is modeled as a parameterized function of states along a trajectory. The proper parameters of a reward function are searched by maximizing the likelihood of the demonstration data.

GAN-GCL integrates IRL and RL by viewing the RL model as a generator and the IRL model as a discriminator that are trained in GAN formulation. The IRL model provides rewards for the RL model. The RL model is trained to gradually shift its sampling distribution to match that of demonstrations. The IRL model is trained to distinguish sampled trajectories from demonstration trajectories, by using the binary cross-entropy loss in Equ. 5.6.

$$L_{irl}(D_\theta) = \mathbb{E}_{\tau\sim p}[-logD_\theta(\tau)] + \mathbb{E}_{\tau\sim q}[-log(1 - D_\theta(\tau))] \tag{5.6}$$

$$D_\theta(\tau) = \frac{exp\{f_\theta(\tau)\}}{exp\{f_\theta(\tau)\} + q(\tau)} \tag{5.7}$$

where D_θ denotes a discriminator model. p and q denote demonstration and sampling distribution respectively. $q(\tau)$ is the probability density estimation of the trajectory from RL-Agent; $f_\theta(\tau)$ denotes the estimation of trajectory rewards, computed by a parameterized model (e.g. a neural network).

The work Adversarial IRL [38] further proposes to replace trajectory τ in Equ. 5.7 with state-action pairs, which makes the training more stable:

$$D_\theta(s, a) = \frac{exp\{f_\theta(s, a)\}}{exp\{f_\theta(s, a)\} + q(a|s)} \tag{5.8}$$

where $f_\theta(s, a)$ denotes the estimation of reward feature on a pair of state and action. We refer to this version of GAN-GCL as State-Action-GAN-GCL (**SA-GAN-GCL**).

5.2 SE Guides Learning by Discovering Underlying Task Logics

Ambiguities commonly exist in communications among people. Due to possible vocabulary differences, when a human teacher attempts to train a robot by showing demonstrations (Robot Learning from Demonstration or LfD), such demonstrations may also suffer from being ambiguous. Maybe from the human's perspective, what he or she intends to demonstrate is clear, but the robot may still make sense of it in a different way. Consider an example illustrated in Fig. 5.1.1: a robot needs to push two components into two different target regions (L1 and L2) that have two different colors (blue and yellow). Imagine a human teleoperates the robot to push the ring object into L1 region in blue, and block object into L2 in yellow. The robot could interpret this demonstration as pushing the block into the yellow region and ring into the blue region or pushing the block into the region L2 and ring into L1.

If humans were in a robot's position, they may ask themselves: What are the features according to the other person that are more relevant to a good decision? After seeing the pushing demonstration, the human may start from a random hypothesize that the object-location relation (e.g. ring_at_L1) is more useful than the object-color relation (e.g. block_at_yellow). Then they would push ring and block always to L1 and L2 respectively, no matter which one is marked in blue or yellow. If the human fails the task once the yellow and blue colors are exchanged (L1 and L2 in yellow and blue respectively), they may consider the object-color relations (e.g. ring_at_blue and block_at_yellow) to be more relevant to a decision. This procedure is essentially self-explanation, which can be used to guide a learner to learn better [131]. According to [131], self-explanations are inferences about relational connections among objects and events, like how procedures cause their effects, and how different structural components can affect a system.

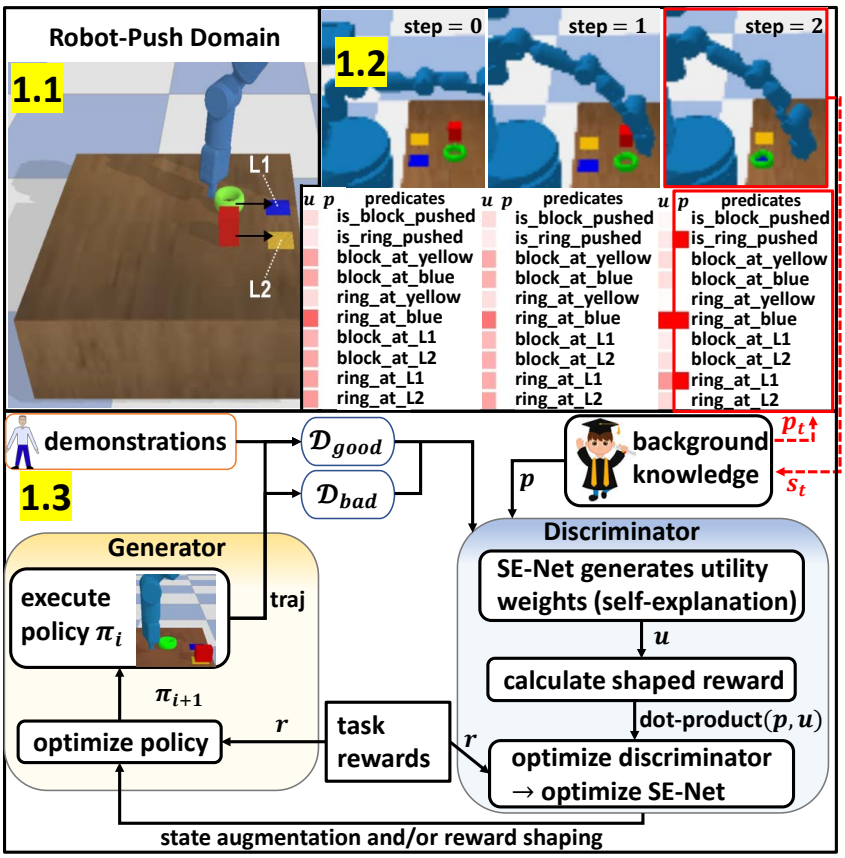


Figure 5.1: Sub-figure (1): the Setting of the Robot-push Task. There are Two Target Regions Indexed By L1 and L2. L1 and L2 are Also Randomly Assigned With Colors Either Blue-yellow or Yellow-blue. To Finish the Task, the Ring and Block Should Be Pushed Into the Blue and Yellow Region Respectively. Sub-figure (2): an Example of a Three-step Robot Execution With Grounded Predicates ($\mathbf{p}$) and Self-explanations ($\mathbf{u}$) Per Step That Is Generated From our Model. Sub-figure (3): The SERLfD Framework That Couples the Learning of Self-explaining (inside the Blue Region) and an RL-Agent (inside the Yellow Region). The Learning of Self-explanation Is Integrated Into a Discriminator. The Policy Can Be Viewed as a Generator. A Robotics Expert Provides Background Knowledge Like Predicates Grounding Procedures for Robots, Which Allow Robots to Disambiguate Demonstrations From Non-experts.

Note that humans do not look at the world at the level of pixels, but could at more abstract cognition levels, like concepts ([69; 137]), factors ([64]), and symbols ([101]). As for robots, since they take human advice (demonstrations), intuitively it would be beneficial if robots do disambiguation reasoning at a similar level to humans so that control-level learning can be guided. Therefore, we assume the robot has some background knowledge like classifiers for extracting important relations that can map continuous states to a set of predefined predicate symbols, which describe various relational connections among objects and/or events. A robotics expert could provide such background knowledge that helps solve a class of tasks (e.g. Pushing).

Therefore, the key insight behind our approach is: **by identifying which object-event relations from each interaction are likely to be more relevant to a decision, an agent can interpret why a trajectory is successful or unsuccessful so that its learning would be guided and improved**.

Reinforcement-**L**earning **f**rom **D**emonstration (**RLfD**) approaches (e.g. [50; 149; 127; 112]) that use demonstrations to accelerate RL training have the benefits of only requiring a few demonstrations. However, traditional RLfD approaches may not be able to handle ambiguous demonstrations efficiently. An optimal trajectory from humans may also suffer from being ambiguous. This could make learners fail the task during training by simply "copying" the demonstration due to varying initial states, which is common in robotic tasks.

To mitigate this issue, we (robotics experts) give robots the aforementioned background knowledge so that robots can understand non-expert demonstrations with human-relevant knowledge. Robots would then need to identify salient relations. We introduce a neural network called self-explanation network (SE-Net) to explicitly output a set of predicate utility weights per step. Each utility weight indicates how utilizable a ground predicate is for interpreting a decision. Therefore, the utility

weights serve as an explanation hypothesis that self-explains which relations are important per step (e.g. the "u" columns in Fig. 5.1.2). Such information could guide RL-Agents to learn better. At the step 2 in the Fig. 5.1.2, while both "ring_at_blue" and "ring_at_L1" are satisfied at this time, weighting them differently can help the agent to connect only useful relations to its decision-making. This also shows that using task-agnostic background knowledge (e.g. human-understandable predicates for a class of tasks) would not remove ambiguity.

However, it is not straightforward to learn to predict proper utility weights for each predicate by using background knowledge. While a human demonstrator provides demonstrations for a specific task, we assume that she/he would not directly tell robots which predicates are important. In other words, we do not have such labels. Inspired by the aforementioned example of how a human may solve a similar problem, one promising direction is to motivate robots to contrastively discover important predicates by distinguishing between successful and unsuccessful trajectories. This requires integrating the ability of self-explaning and that of practising with the current self-explanation hypothesis. To this end, we propose a **S**elf-**E**xplaination for **R**einforcement-**L**earning **f**rom **D**emonstration (**SERLfD**) framework based on Generative Adversarial Inverse Reinforcement Learning (GAN-IRL) methods [35; 38]. In the GAN-IRL works, the learning of a non-linear reward predictor and an RL-Agent can guide each other in a generative-adversarial fashion. In our approach, as illustrated in Fig. 5.1.3, we modify the reward predictor in GAN-IRL by inserting into it a self-explanation network (SE-Net) that generates predicate utilities. While the whole reward predictor can be trained as usual via a discriminator loss, we essentially use the predicted utility weights (i.e. self-explanations) to guide the learning of RL-Agents.

Our main contribution is the SERLfD framework that contrastively and iteratively

learns to use background knowledge to self-explain which high-level predicates are task-relevant and meanwhile perform the task. Our SERLfD algorithm is essentially a modification of traditional GAN-IRL methods that were used for Imitation Learning (or LfD). Our SERLfD algorithm combines the benefits of both worlds (RLfD and GAN-IRL). We extensively evaluate our SERLfD framework with multiple candidate RL-Agents in four continuous robotics domains and one discrete Pacman domain. Our evaluation demonstrates that SERLfD clearly outperforms traditional RLfD and GAN-IRL methods with better learning stability and higher scores even in challenging continuous control domains. To the best of our knowledge, our work opens the direction of learning to self-explain potentially ambiguous demonstrations to support RLfD.

5.3 Problem

Grounded Predicates Values

A predicate is a logical formulation that describes a relationship among several objects. The objects that a predicate concerns would form the arguments of the predicate. Take for instance a predicate "pushed(obj)" in the aforementioned robot-pushing domain, "obj" is an argument that can be grounded with "block" or "ring". When the argument is grounded, the predicate becomes a grounded predicate, e.g. "pushed(block)". In our work, we use a **binary predicate variable** to represent if a grounded predicate is satisfied or not. For example, "pushed(block)" can be represented by a binary predicate variable "is_block_pushed". When it is satisfied, we assign this binary predicate variable with value 1, otherwise -1. We may have a set of grounded predicates (and thus a set of binary predicate variables) to describe the most important relations according to domain knowledge. **Grounded predicate**

values at the current step are the values of all binary predicate variables obtained by processing the current environment state. Since common functions can be reused to ground certain groups of predicates, and a set of predicates could be shared across multiple tasks, it is not expensive to provide agents with such predicate knowledge.

Self-Explanation Generation

We model the problem of generating a self-explanation as finding a set of **predicate utility weights** that match the set of binary predicate variables. A predicate utility weight serves as a quantitative estimation of how utilizable the corresponding predicate variable is to interpret why making a decision is promising, given a state. Finding utility weights (which can be either positive or negative) for all predicate variables may highlight some predicate variables and play down the others. This way, we approximately build a connection between a subset of relations and a decision, since a predicate is an abstracted representation of a relation. Even in extreme cases that all predicates have satisfied/unsatisfied groundings, probably only a subset of them are recognized to be useful. Then the utility weights for those predicates would probably have larger absolute values. In our work, we use a neural network to generate a set of utility weights from a state, which will be explained in Sec. 5.4.2.

Self-Explaination for RL from Demonstration (SERLfD)

Solving a SERLfD problem is similar to solving an RLfD problem as in [149; 50; 127]: They use demonstrations to initialize a replay buffer and bootstrap the learning. During the later training stages, whether demonstration experiences are still used depends on how good the learning is. What distinguishes SERLfD from RLfD are as follows: 1) Our environment provides both states and grounded predicate values with an internal state processor; 2) The input for RL-Agent is a concatenation of state,

grounded predicate values, and utility weights; 3) SERLfD would use a self-explainer submodule to generate explanations (predicate utility weights) for each state. The generated explanation would then be used to guide the learning of RL-Agent.

5.4 SERLfD Framework and Algorithm

In this section, we introduce our SERLfD framework (Fig. 5.1.3) which couples two objectives together. One is to learn self-explanations that identify task-relevant relations by distinguishing between successful and unsuccessful trajectories. The other is to encourage RL-Agents to use self-explanations so that RL training can be improved. We introduce a self-explanation network (SE-Net) that maps a state to a set of utility weights $\boldsymbol{u}$ (self-explanation) that corresponds to a set of grounded predicates values $\boldsymbol{p}$. The predicates and their recognition classifiers can be viewed as background knowledge that a robotics expert offered for helping a robot to handle a class of tasks. We also propose to improve an RLfD agent with self-explanations by either 1) state augmentation, i.e. $\langle s \rangle$ becomes $\langle s, p, u \rangle$; or 2) reward augmentation, i.e. $\hat{r}_\theta(s, u, a, r(s, a))$ where $r(s, a)$ is task reward function that comes from an RL environment. The task reward also serves as an indicator that decides whether a trajectory is successful or not. Since we rely on task rewards for learning both SE-Net and RL-Agent, we do not break RLfD assumptions that demonstrations are used to guide RL to maximize accumulative task rewards. Our RL-Agents can be any deep RL algorithm that is adjusted to learn from demonstrations a policy that optimizes accumulative task rewards, e.g. [50; 149]. To contrastively learn the SE-Net, we merge the objective of learning to self-explain into the learning of GAN-IRL frameworks ([35; 38]). The discriminators in GAN-IRL frameworks are trained by distinguishing between two different distributions of trajectories: demonstrations or generated samples. Unlike GAN-IRL, the discriminator in SERLfD includes the SE-

Net and is trained differently by distinguishing between successful and unsuccessful trajectories. We will explain what are grounded predicate values (Sec. 5.4.1), how to learn to "self-explain" (Sec. 5.4.2), how RLfD learning could be benefited from self-explanations while RL-Agents also adapts the learning of SE-Nets further by providing new samples (Sec. 5.4.3).

5.4.1 Grounded Predicate Values

A predicate is a logical formulation that describes a relationship among several objects. The objects that a predicate concerns would form the arguments of the predicate. Take for instance a predicate "pushed(obj)" in the aforementioned robot-pushing domain, "obj" is an argument that can be grounded with "block" or "ring". When the argument is grounded, the predicate becomes a grounded predicate, e.g. "pushed(block)". In our work, we use a **binary predicate variable** to represent if a grounded predicate is satisfied or not. For example, "pushed(block)" can be represented by a binary predicate variable "is_block_pushed". When it is satisfied, we assign this binary predicate variable with a value of 1, otherwise -1. We may have a set of grounded predicates (and thus a set of binary predicate variables), according to background knowledge, to describe the most important relations. We assume that the classifiers for detecting the satisfaction of such predicates are available. The rapidly growing computer vision community is already able to provide high-quality visual recognizers that detect objects and relations (e.g. through scene graph analysis [157]). There are also works that improve RL by leveraging ground-truth symbolic high-level knowledge, e.g. [81; 147; 21; 156].

5.4.2 Training the SE-Net in a Discriminator

The input to the Discriminator (in Fig. 5.1.3) is obtained by sampling a batch of experiences and grounded predicate values from buffers $\mathcal{D}_{good}$ and $\mathcal{D}_{bad}$. With an experience, SE-Net takes a state s as input and generates a set of **predicate utility weights** $\boldsymbol{u}$: $u = SE\text{-}Net(s)$. A predicate utility weight serves as a quantitative estimation of how utilizable the corresponding predicate variable is to interpret why making a decision is promising, given a state. The values in $\boldsymbol{u}$ may highlight some predicate variables and play down the others. This way, we identify salient relations for a decision, since a predicate is an abstracted representation of a relation. Even in extreme cases that all predicates have satisfied/unsatisfied groundings, probably only a subset of them are recognized to be useful. Then the utility weights for those predicates would probably have larger absolute values.

Self-explaining can be seen as finding a set $\boldsymbol{u}$ of utility weights. The number of output utility weights per step should match the number of grounded predicate values. The neural network architecture of SE-Net is flexible and should match the state-space. In some of our experiments, our state-space consists of features like object poses. Then SE-Net could be implemented by using a multilayer perceptron.

By taking the dot-product of $\boldsymbol{u}$ and grounded predicate values, we can obtain a scaler value $h_\theta(s)$ that we would use later:

$$h_\theta(s, u) = \sum_{i=0}^{k-1} u_i \cdot \mathbb{P}(s)_i \tag{5.9}$$

where $\mathbb{P}(s)$ denotes the grounded predicate values extracted from state s, and k denotes the number of predicates. We explain how we obtain the function $\mathbb{P}$ in the **Sec. 2** of linked **supplemental manuscript** (note1).

Recall that GAN-GCL [35] and SA-GAN-GCL [38] train IRL and RL models together as a GAN (a coupling of discriminator and generator). Traditionally, the

IRL model is trained by differentiating demonstration and sampled trajectories. The IRL model (discriminator) also provides rewards for training the RL-Agent to **imitate** the demonstrations. However, solving a SERLfD problem is closer to solving an RLfD problem as in [149; 50; 127]: They use demonstrations to initialize a replay buffer and bootstrap the learning **of maximizing task rewards.**

Now an important question is how can we integrate the learning of the SE-Net in a discriminator and an RL-Agent. Inspired by the difference between RLfD that requires task rewards and GAN-GCL based Imitation Learning that only needs a discriminator (discussed above Eq. 5.6), we decide to use the predicted utility weights $\boldsymbol{u}$ to shape the task rewards. The value $h_\theta(s)$ in Eq. 5.9 is essentially a reward shaping term that rely on both SE-Net and background knowledge (predicates and $\mathbb{P}(s)$). Then we can compute the prediction of shaped rewards $\hat{r}$ as:

$$\begin{aligned} &\hat{r}_\theta(s_t, u_t, a_t, s_{t+1}, u_{t+1}, r) = \\ &r(s_t, a_t) + h_\theta(s_{t+1}, u_{t+1}) - h_\theta(s_t, u_t) \end{aligned} \tag{5.10}$$

where $\hat{r}_\theta()$ models the prediction of shaped rewards $\hat{r}(s_t, a_t)$, and $h_\theta(s)$ is computed in Eq. 5.9. $r(s_t, a_t)$ is the task rewards provided by an RL environment. The formulation of Eq. 5.10 is based on Eq. 5.5 and shares similarities with the prediction of shaped rewards in work [38]. However, different from [38], we include task reward r to train the agent to accomplish the task instead of merely imitating. In the rest of this paper, we **shorten the** $\hat{r}_\theta()$ **with long lists of inputs to** $\hat{r}_\theta(s, u, a, r)$.

Since our self-explanation $\boldsymbol{u}$ only "augments" the task rewards, our discriminator is trained to distinguish between successful and unsuccessful trajectories. In the traditional GAN-IRL works, the task reward is missing. All demonstrations are labeled as good and all sampled trajectories are labeled as bad. But incorporating task rewards can better motivate Self-Explainer to learn to discriminatively recognize which predicates are more useful for solving a task. Sampled trajectories that can

accomplish a task, in our problem, should be treated the same as demonstration trajectories. Remember in Fig. 5.1.3, we store a sampled trajectory into either $\mathcal{D}_{good}$ or $\mathcal{D}_{bad}$ depending on if it accomplishes a task. The discriminator loss L_{SE} for training the SE-Net can be formulated as:

$$L_{SE} = \mathbb{E}_{(s,a)\sim\mathcal{D}_{good}}[-logD(\hat{r}_\theta(s,u,a,r))] + \mathbb{E}_{(s,a)\sim\mathcal{D}_{bad}}[-log(1 - D(\hat{r}_\theta(s,u,a,r)))]$$

where $u = SE\text{-}Net(s)$, the discriminator function $D()$ is formulated in Eq. 5.8, and $\hat{r}_\theta(s,u,a,r)$ in Eq. 5.10.

5.4.3 Improving RL-Agent (Generator) with Self-Explanation

The yellow area in Fig. 5.1.3 depicts how we train the RL-Agent with the help of predicted self-explanations. Note that the SE-Net learns a complex non-linear function to "explain" a state. SE-Nets also provide more detailed guidance (predicate utilities) rather than merely a numerical reward prediction. In our work, we propose two ways for an RL-Agent to take advantage of SE-Nets. One way is to augment RL states by concatenating them with grounded predicate values and predicted utility weights (self-explanation): $\langle s \rangle$ becomes $\langle s, p, u \rangle$. Then policy network in RL-Agents becomes: $a \sim \pi_\theta(s,p,u)$. Another way is to estimate a shaped reward $\hat{r}_\theta(s,u,a,r)$ by using Eq. 5.9 and 5.10. The former provides more detailed guidance whereas the (shaped) rewards are more direct learning signals for RL-Agents. Intuitively, a good explanation provides better guidance for the RL-Agent to sample a trajectory that in turn makes the discriminator harder to distinguish, which adapts the SE-Net further.

The SERLfD learning is summarized in **Algorithm. 1**.

Algorithm 5 The SERLfD Learning Algorithm

INPUT: A dataset of human demonstrations, an environment with reward function r, state space S, action space A, and all hyper-parameters

Initialize the weights of an RL-Agent and a Self-Explainer

Initialize buffers $\mathcal{D}_{good}$ and $\mathcal{D}_{bad}$ for training the Self-Explainer and D_{RL} for RL from Demonstrations as in [50; 149]

Store expert experiences into $\mathcal{D}_{good}$ and $\mathcal{D}_{RL}$. Pretrain the RL-Agent with experiences sampled from $\mathcal{D}_{RL}$

Sample K trajectories with a random policy and add them to D_{RL}. Also add successful and unsuccessful trajectories to $\mathcal{D}_{good}$ and $\mathcal{D}_{bad}$ respectively.

for $episode = 1; episode \leq N; episode++$ **do**

  Sample experiences (including grounded predicate values) from $\mathcal{D}_{good}$ and $\mathcal{D}_{bad}$ ▷ **Train Self-Explainer**

  Compute utility weights u and shaped reward prediction $\hat{r}_\theta(s_t, u_t, a_t, s_{t+1}, u_{t+1}, r)$ by using SE-Net, Eqs. 4 and 5

  Update the SE-Net via binary cross entropy loss L_{SE} to distinguish successful experiences from unsuccessful experiences

  Sample experiences with grounded predicate values from $\mathcal{D}_{RL}$ ▷ **Train RL-Agent**

  Run SE-Net on sampled states to obtain utility weights u

  Augment input states with grounded predicate values and utility weight values

  Augment rewards with predicted shaped reward by using Eq. 5

  Update RL-Agent with the augmented experiences

  Use RL-Agent to sample a new trajectory and add it to D_{RL}. If the trajectory is successful, it would also be added to $\mathcal{D}_{good}$. Otherwise, it would also be added to $\mathcal{D}_{bad}$ ▷ **Sample a new trajectory**

end for

return a trained RL-Agent and SE-Net

5.5 Evaluation

Since self-explanations should play a general role in improving an RL-Agent that is trained with ambiguous demonstrations, we evaluate our SERLfD framework in multiple domains and use different candidate deep RL models as the RL-Agent. We evaluate the RL learning performance in this section and the predicted self-explanations in our **supplemental video** (note1). We design our evaluation to answer the following questions: 1) Can SERLfD outperform RLfD? 2) Which of the two ways of using self-explanation to guide an RL-Agent (state or reward augmentation) is more helpful? 3) What could happen if the self-explanation is used to define the entire reward rather than just the reward shaping? 4) Do self-explanations play a general role in supporting an RL-Agent to learn from ambiguous demonstrations or self-explanations are only effective for certain of the RL models? 5) Since our SERLfD combines the benefits of RLfD and GAN-IRL, does our SERLfD outperform a state-of-the-art GAN-IRL as an Imitation Learning method? 6) Does SERLfD help in both continuous and discrete domains?

To answer 1, we compare the performance of an RLfD model and the same one supported by SE-Net. To answer 2, we do extensive studies of how RL agents can use self-explanations by removing the predicate utility weights from the input of an RL-Agent (RLfD+SE+**nu**), or by removing the reward shaping terms (RLfD+SE+**nrs**). To answer 3, we remove task rewards and only use the dot-product of predicate utility weights and predicate values as rewards to train RL agents (RLfD+SE+**ntr**). To answer 4, we investigate a diverse set of RL-Agents. To answer 5, we compare SERLfD with the SA-GAN-GCL (for Imitation Learning) proposed in [38]. To answer 6, we evaluate our models in three continuous robotic control domains and one discrete Pacman domain.

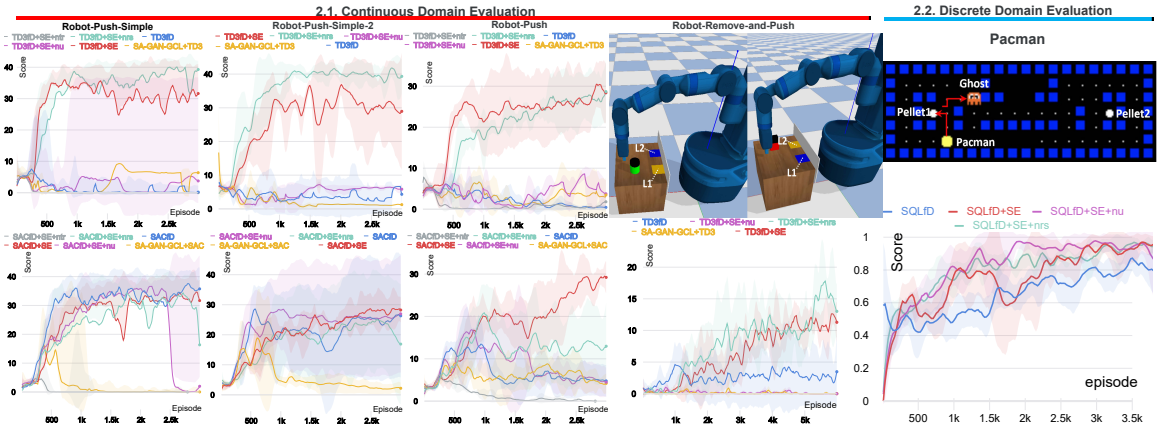


Figure 5.2: 2.1. Learning Curves of Training the Baseline RLfD Agents (TD3fD/SACfD), RLfD With SE-Nets (RLfD+SE), the Extensive Studies of how RLfD Can Use Self-explanations (RLfD+SE+nu and RLfD+SE+nrs), Using the Self-explanation to Form the Entire Rewards for RL-Agents (RLfD+SE+ntr), and an Imitation Learning Agent Built By Using RL in the Original SA-GAN-GCL Framework [38]; And 2.2. The Pacman Domain and the Learning Curves With the RL-Agent SQLfD. For Each Curve, We Run Three Times of Each Algorithm and Report the Mean and Standard-deviation, Which are Plotted in the Bold and Lighter Color Region Respectively. Y-axis Values are Scores That Each Is Measured as an Average Over 100 Episodes. X-axis Values are Episodes.

5.5.1 Experiments in Continuous Domain

We start by evaluating SERLfD in a **Robot-Push** domain (as described in Fig. 5.1) that a continuous control model would need to be learned. We used a Fetch Mobile Manipulator [152] that has a 7-DoF arm in PyBullet simulator [26]. We also fixed its mobile base in experiments. In this Robot-Push domain, there are two target regions indexed by L1 and L2. **Note that this indexing is static**. By saying a "region", we mean a square that each side is approximately 0.1m. The region L1

and L2 are assigned with either blue-yellow or yellow-blue colors. The predicate variables are: {is_block_pushed, is_ring_pushed, block_at_yellow, block_at_blue, ring_at_yellow, ring_at_blue, block_at_L1, block_at_L2, ring_at_L1, ring_at_L2}. The task is to push the block and ring into yellow and blue regions respectively, no matter which is L1 or L2. Our design of state-space follows the FetchPush-v0 environment made by OpenAI-Gym [1] . The state-space consists of the positions and orientations of the objects in domain w.r.t the world frame. Such objects include the end-effector, ring, block, regions in blue and yellow colors, and regions L1 and L2. The action space is defined as a 4-tuple: [translation_x, translation_y, translation_z, yaw_angle] of the end-effector w.r.t world frame. Once the block is pushed to the yellow region and ring to the blue region, the robot finishes a task. The task reward function is sparse: In most situations, $r(s, a)$ is zero; If robot pushes an object towards its target location to be δd closer, it receives a reward of $100 * \delta d$; If either ring or block is pushed into its target region, the robot receives a reward of 25; If both ring and block are pushed into their target regions, the robot receives a reward of 50.

We also designed two simplified versions of the aforementioned Robot-Push domain, named **Robot-Push-Simple** and **Robot-Push-Simple-2**. In the Robot-Push-Simple domain, the predicate variables are: {is_block_pushed, is_ring_pushed, block_at_yellow, block_at_blue, ring_at_yellow, ring_at_blue}. The region L1 and L2 are always in blue and yellow. Other details are the same as the Robot-Push domain. The Robot-Push-Simple-2 domain is identical to the Robot-Push-Simple domain except for that it has a larger predicate set that is the same as the Robot-Push domain.

We use two state-of-the-art RLfD baselines for continuous control tasks: Twin-

[1] `https://gym.openai.com/envs/FetchPush-v0/`

Delayed DDPG [39] from Demonstrations (TD3fD), and Soft-Actor Critic [49] from Demonstrations (SACfD). To collect demonstrations, we use a keyboard to input control commands. We collected 8 trajectories (averagely 17 steps each) for the Robot-Push-Simple and Robot-Push-Simple-2 domain and 15 trajectories (averagely 19 steps each) for the Robot-Push domain.

Robot-Push-Simple, Robot-Push-Simple-2, and Robot-Push

The results are reported in Fig. 5.2.1. We extensively study what is the best way for an RL-Agent to use self-explanations by introducing the models RLfD+SE+**nrs** and RLfD+SE+**nu** along with our main model RLfD+SE. Please note that all of the three models that use our self-explainer – either for reward shaping, state augmentation, or both – are our models that require a SERLfD framework and SE-Net. The results present multiple interesting findings. First, the RL-Agents with SE-Nets generally can be trained more stably and achieve higher scores within a shorter time. One exception is when we use SACfD as the baseline RL-Agent in Robot-Push-Simple and Robot-Push-Simple-2. This is because the two domains have a lower degree of ambiguity. The entropy-driven exploration in SACfD can find the optimal policy without the help of self-explanations. Second, the results show that using self-explanations to augment states is better than constructing shaped rewards with self-explanations. The reason could be 1) the state augmentation provides detailed information of self-explanation to RL-Agents, and 2) due to RL-Agents' explorations, the reward shaping that contains self-explanations would not be fully used at the beginning stage of training. We also evaluate an incorrect way of using self-explanation to form the entire rewards (RLfD+SE+**ntr**). Theoretically, RLfD assumes the task/extrinsic reward is available. Demonstrations are used to guide RL agents to find a policy that maximizes the accumulative extrinsic reward faster. Using the self-explanation

to form the entire reward breaks this assumption of RLfD. Third, the domain Robot-Push is harder than Robot-Push-Simple-2 but they have the same predicates. Our results for the two domains show that the RLfD agents with SE-Nets can maintain good performance even for harder tasks.

Robot-Remove-and-Push

Based on the aforementioned results, we evaluate some interesting models in this more complex robot domain. There are two target regions indexed with L1 and L2. The region L1 and L2 are assigned with either blue-yellow or yellow-blue colors. L1 and L2 are fixed whereas blue and yellow are exchangeable. In each episode, either a block or a cylinder would show up. Both have a black cover at the top. If their initial poses are on the left side of the table, the robot needs to push them to their target regions **with the black cover removed**. The target regions for the block and cylinder are blue and yellow regions respectively. Robots only get a reward of $+50$ when they accomplish the task. This domain supports a more complex task of 20 predicates (Sec. IV in the supplemental material in note1). We collected 16 demonstrations (averagely 5 steps each). We use TD3fD as a representative of RLfD models. We report the results in the last column of Fig. 5.2.1 which shows clear benefits from learning self-explainers.

5.5.2 Experiments in Discrete Domain

We report our results in a discrete **Pacman** Domain (Fig. 5.2.2) to demonstrate if self-explanation is helpful with discrete state and action spaces. After taking the power pellet, the Pacman should try to eat ghosts within a fixed amount of time. Once the Pacman eats all of the randomly-wandering ghosts it completes the task. Then it gets a reward of 1 and the current episode ends, otherwise, the reward is

0. Two predicate variables, {ghost_nearby, eat_capsule}, are provided to describe whether the ghost is close to the Pacman and whether the Pacman has eaten a pellet. The ambiguity in this domain lies in the proper time to eat a pellet. In the demonstrations (e.g. the red trajectory in Fig. 5.2.2), the agent rushed to eat the pellet because the ghost was coincidentally nearby. But normally, the agent should wait to eat the pellet until the ghost approaches. Our RL-Agent is Soft Q-Learning [48] from Demonstration (SQLfD). We collected 5 demonstrations (averagely 18 steps each). In our training, each episode has at most 2000 steps. From the results, we can conclude that the RL-Agents with SE-Nets perform better.

5.6 Related Work

5.6.1 Deep Reinforcement Learning from Demonstrations

Many works have investigated the benefits of using RLfD frameworks. The work [127] uses states in demonstrations as starting points to train DRL with short-term interactions. As such, explorations only happen in a local region around a good state from demonstrations. [50] and [149] propose to use demonstrations to initialize the replay buffer so that the training of a deep Q-Net ([50]) or a deep deterministic policy gradient (DDPG) network ([149]) can be benefited. [28; 112; 108] propose to directly use demonstrations to initialize neural network parameters by pretraining it with an imitation learning objective. Besides pretraining a DRL model, imitation learning can also be used to construct an auxiliary imitation learning loss ([149; 93]). The work [41] addresses the problem of DRL from imperfect (noisy and corrupted) demonstrations. [41] relies on reward information to perform a Q-function normalization over actions. However, learning from ambiguous demonstrations is an orthogonal issue: Such demonstrations could lead to the highest reward, may not necessarily be

corrupted, but confuses the learner.

5.6.2 Imitation Learning from Ambiguous Demonstrations

So far only a few works have attempted to do robot LfD from ambiguous demonstrations as in [14; 11; 89; 30]. [14] and [11] model demonstration ambiguity as the intention differences of human and robot. [14] proposes to let a human teacher and robot learner gradually improve the learning difficulty and quality. By explicitly modeling human's intention and belief in a Bayesian inference framework, the robot can discover conflicts, query humans, and learn better. [11] models the ambiguity as a joint hypothesis space of multiple categories of concepts (e.g. colors, shapes). To disambiguate the demonstration is to reduce the hypothesis space. [11] then proposes a concept learning approach to reduce the hypothesis space by using new demonstrations step by step. [89] models the demonstration ambiguity as a difference in demonstrated actions at the same (or similar) state. To disambiguate demonstrations, [89] adopts clustering algorithms and proposes a two-level clustering approach to simultaneously categorize similar situations and ambiguous actions in each situation. Another work [16] proposes to use Bayesian optimization to infer reward uncertainty learned by an IRL algorithm, which has the potential of doing imitation learning from ambiguous demonstrations. However, [16] mainly focuses on the IRL side and evaluates in simple domains. The work [30] addresses distribution drift problems [124; 125] in imitation learning by avoiding causal misidentification caused by using ambiguous demonstrations. While [30] touches on the RL, it is not an RLfD approach because the task (environment) rewards are not used. In contrast, according to [112] the RLfD approaches and our work can naturally handle distribution drift problems since we directly use task rewards.

5.7 Concluding Remarks

In this work, we propose the **S**elf-**E**xplaination for **R**einforcement-**L**earning **f**rom **D**emonstration (**SERLfD**) framework for allowing RLfD agents to efficiently use even ambiguous demonstrations by doing self-explanations. Our extensive evaluation shows appealing benefits of training RLfD from learning self-explanations in one discrete Pacman and three challenging robotics control domains. The experimental advantage also suggests that our SERLfD could lead to compelling practical applications since ambiguous demonstrations could frequently happen when robots cohabitate with non-practitioner humans in real-worlds.

5.7.1 Significance

In this work, we open the direction of using self-explanation to help reinforcement learning from human demonstrations. One may continue this direction by:

1) Improving the self-explanation mechanisms that can fit for more and more challenging settings;

2) Applying the self-explanation mechanism to help address other interesting problems;

3) Studying the broader version of introspection for robot learning – that goes beyond the self-explanation.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Summaries of Contributions

This dissertation aims at achieving human-level intelligence by tightly coupling different levels of cognitive functions like perception, acting, planning, and introspection. This dissertation explores novel directions for helping different levels of cognitive functions learn and adapt together based on deep neural networks. The dissertation can be divided into three main threads: perception-acting coupling, perception-planning coupling, and self-explanation-guided learning.

I started out by studying **perception-acting coupling** – how the learning of perception and that of acting could support each other in a tightly coupled fashion. The perception-acting coupling is closely relevant to the concept of affordance which describes the correlations among objects, actions, and effects. Traditional grasping affordance prediction works typically predict rigorous forms of affordances so that an external motion planner can accept it to execute and achieve certain effects like picking a mug up. An important cluster of works along this direction includes learning affordance from demonstrations [29; 31] which avoid the labor-intensive affordance labeling collection. My work [164], however, is the first to combine two important directions together: learning (grasping) affordances from demonstrations and deep learning based imitation learning. This enables robots not only to learn to imitate humans at the level of trajectories but also imitate their tacit knowledge of affordances – which achieves **affordance-aware imitation learning**.

While the above direction of perception-acting coupling makes the perception

"more conscious", the overall learning process still lacks "long-term thinking" which is crucial in a cognitive agent [68]. Thus, my second thread of research focuses on a higher level of cognitive learning – the **perception-planning coupling**. My research [166; 176] focuses on plan recognition problems that can be reduced from planning problems. Different from traditional works that treat perception and plan recognition modules as the separate parts of a pipeline, my research introduces a learnable perception-planning interface that allows a high-level plan recognition module to effectively utilize even erroneous low-level perception outputs [166]. While most perception-planning-or-plan-recognition works consider the flow from perception to plan recognition, I further ask the question of whether we can do the opposite of using plan recognition to improve perception [168]? My idea becomes part of a successful ONR (Office of Naval Research) proposal.

Finally, to pursue human-level intelligence for robots, I also research if a cognitive learning agent may also introspect (or **self-explain**) what it has experienced so far, either from itself or from others. Likewise, humans may also frequently monitor their learning and draw lessons from failures and others' successes. To this end, I explore the possibility of motivating cognitive agents to learn how to **introspect** human demonstrations and agents' own experiences, i.e. accomplishments and failures, to gain useful insights. By internally making sense of the past experiences, advanced cognitive agents could have their learning of other cognitive functions guided and improved. My research in this thread [165] promises to open a new direction for the robot learning community – self-explanation guided robot learning from human demonstrations.

As many research endeavors that illuminate an unexplored region in darkness, the research presented here may also suggest many open questions and unaddressed challenges. Many of my presented works leverage human advice (e.g. demonstrations)

to speed up the learning of cognitive agents as discussed in Chapter 2 and 5. Being able to incorporate human advice also allows human users to *customize* learning agents. One fundamental question is what should be the best form of human advice and what could be the best way of incorporating human advice? I will provide some partial answer towards the open question in Section 6.2 based on our Blue Sky paper [58] in Section. 6.2.

Furthermore, one may discover interesting directions for future research by exploring potential connections among the three threads – perception-acting coupling, perception-planning coupling, and self-explanation guided learning. In this dissertation, I will point out some connections among the three threads that could lead to interesting future works in Section 6.3.

6.2 Cognitive Learning with Humans Advice

Using human advice would greatly improve the learning efficiency of agents. To achieve this, one naive way is to ask humans to figure out how to understand robots' own (internal) representations. In this sense, there is no need to maintain an interface between humans and learning agents. However, Human-AI systems should be designed for the benefits of humans. Therefore, agents should maintain a communication channel so that humans can use to advice agents.

Given the popularity of learning to make decisions from pixels, it seems to be more straightforward to ask humans and learning agents to communicate at the level of raw data like images, videos, or trajectories. Communicating at the level of raw data helps agents and humans convey each other tacit knowledge like how to grasp a mug as in Chapter 2. However, this would make it inappropriate for tasks that involve more explicit knowledge due to high cognitive load for humans. One great example is my self-explanation paper [165] presented in Chapter 5, which shows that in more

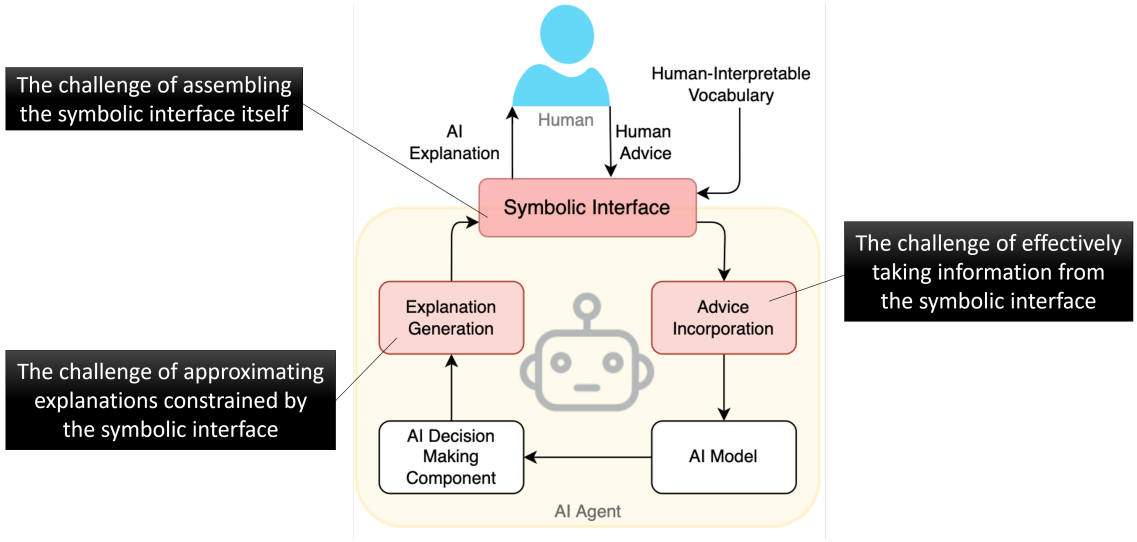


Figure 6.1: Overall Architecture of an AI System Exposing a Symbolic Interface to a Human User, Enabling the AI Agent to Provide Explanations to its Decisions as Well as Accept Guidance/preferences From the Human in the Form of Advice.

complex domains, human demonstrations can easily convey ambiguous information to robots and hinder robots' learning. Indeed, we humans do not communicate with pixels or videos in the most of the time! Instead, we have developed our languages, or more fundamentally, symbols and their combinations.

As illustrated in Figure. 6.1, [58] argues that it is important for robots to maintain and use a symbolic interface to make them more explainable (explain to humans why robots make a decision) and advisable (learn from human advice from humans' perspectives) – even if robots' core computation modules (e.g. the AI model and AI Decision Making Component in Figure. 6.1) are inscrutable to humans.

One may find it straightforward to imagine how convenient it could be if robots can clearly explain their decision-making to human users in human-understandable symbols (thorough a symbolic interface). However, it could be more subtle to think about the benefits that robots can gain if they learn from human advice thorough

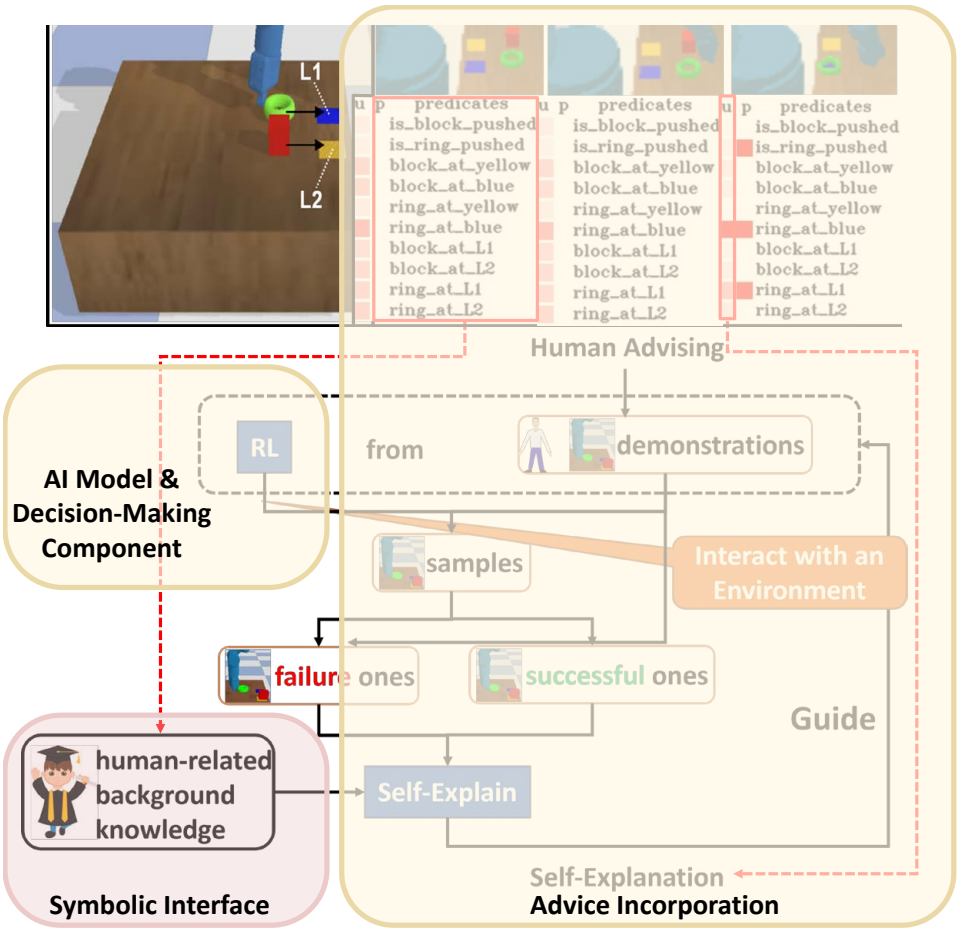


Figure 6.2: Interpreting Human Demonstrations in the Context of the Symbolic Interface (c.f. [165]

the symbolic interface, which allows robots to think about human advice from humans' perspectives. One everyday example is that when you see someone who is gesturing to you, you would deliberately think about the symbolic meaning behind the gesture with the context of the person. The self-explanation guided robot learning work [165] explained in Chapter 5 also belong to this direction of making robots more advisable by having them learn human demonstrations thorough a symbolic interface. As described in Figure. 6.2, the set of predicates and their detection functions (human-related background knowledge) constructs a symbolic interface be-

tween robot learners and human teachers. Robots learns to better incorporate human advice by introspecting and self-explaining which predicates are task-relevant. The predicted self-explanations at each step help guide the AI model and Decision-Making component.

6.3 Future Directions

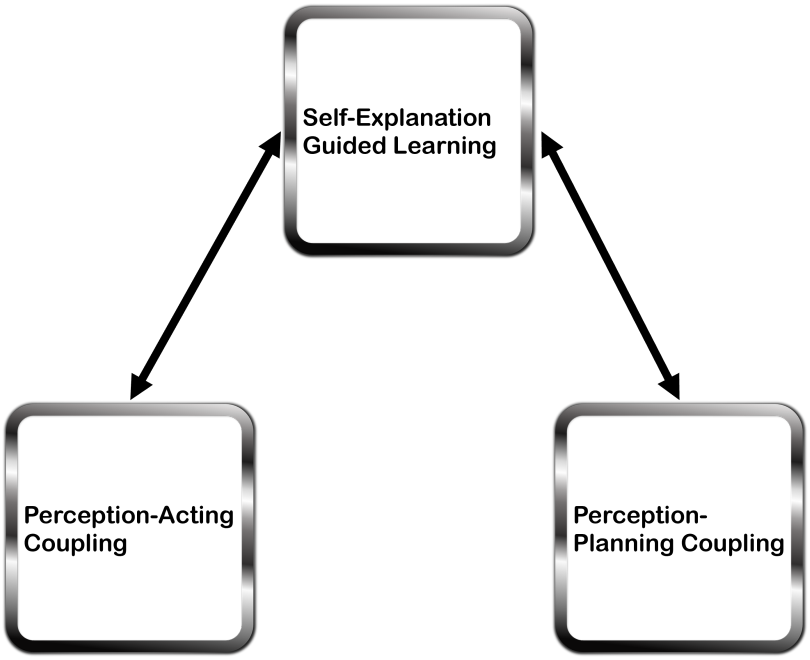


Figure 6.3: The Section 6.3 Would Focus on Exploring Novel Connections Between the Self-Explanation Guided Learning Thread and the Other Two Threads: 1) Perception-Acting Coupling and 2) Perception-Planning Coupling.

Based on the discussion above, I focus on exploring new connections between the self-explanation thread and the other two threads as illustrated in Figure. 6.3. This is because combining self-explanation and deep learning could achieve human-level intelligence, yet is quite under-explored in the robot learning community. A paper published in 1999 ([27]) proposes to use self-explanation to enhance the learning of

an agent. Unlike [27], the research presented in Chapter 5 views the self-explanation from a more general perspective of connecting cognitive psychology and modern deep neural networks for robotics.

Endowing robots that learn from teachers with the human-like self-explanation mechanism brings up several significant challenges: 1) the challenge of defining the form of teachers' advice and formulating its incorporation – either at the level of symbolic representations or raw data (space-time signal tubes or STST in short) [58]; 2) the challenge of designing the self-explanation mechanism that can be tightly coupled into robots' learning algorithms; 3) the challenge of using self-explanation to improve lower-level cognitive components, like perception, planning and performance components, as a whole based on deep neural networks. Solving these challenges might be achieved by exploring potential connections among the three research threads described in Figure. 6.3. The connections between self-explanation guided learning thread and perception-planning coupling thread hints at the need of maintaining symbolic human-related background knowledge that benefits for agents to learn tasks with more explicit knowledge. The symbolic background knowledge would be in the format of models. On the other hand, the connections between self-explanation guided learning thread and perception-acting coupling thread help address the challenges that emerge when cognitive learning agents learn tasks with more tacit knowledge.

6.3.1 Self-Explanation-guided Learning for Tasks that Require Explicit Knowledge

When robots learn from humans the tasks that might need both explicit and tacit task knowledge, robots might need a symbolic interface to help them understand human advice from humans' perspectives as discussed in Section 6.2. The work presented in Chapter 5 demonstrates the benefits of the self-explanation mechanism in supporting a reinforcement learning from demonstrations (RLfD) agent efficiently learn from

ambiguous demonstrations. That said, the learning of self-explanation in Chapter 5 is essentially model-free (without using any formal action or dynamics models). However, note that the symbolic interface that conveys human-related background knowledge can be either model-free (there is no syntactic and semantic constraints over symbols), or be a more systematical organization of symbols, e.g. symbolic or causal models – therefore revealing connections between the self-explanation guided learning thread and the perception-planning coupling thread. One promising future direction, therefore, is to leverage the works that learn and use shallow-domain planning models (Chapter 3 and 4) to develop novel model-based self-explanation mechanisms which output self-explanation hypothesis as guiding signals for learning. This would eventually further improve learning efficiency and generalizability.

6.3.2 Self-Explanation-guided Learning for Tasks that Require Tacit Knowledge

Unlike the previous research thread, cognitive learning agents sometimes need to learn from humans those tasks that require much more tacit knowledge (e.g. Fig. 2.1). Consequently, it might be difficult and unnecessary to develop symbolic representations to help robots learn from human advice. One of the promising direction for future works is to use self-explanation to help learn tacit tasks like Chapter 2 – that is, to explore novel connections between the self-explanation guided learning thread and the perception-acting coupling thread in Figure. 6.3. Note that one strict assumption that we have made in Chapter 2 is that humans demonstrations cover all required tacit affordance knowledge. However, it mostly will not be the case when robot users are non-practitioners. In such cases, robots need to explore. Self-explanation could help cognitive learning agents gain useful insights from their past experiences to guide the exploration.

REFERENCES

[1] Abidi, B. R., N. R. Aragam, Y. Yao and M. A. Abidi, "Survey and analysis of multimodal sensor planning and integration for wide area surveillance", ACM Comput. Surv. **41**, 1, 7:1–7:36 (2009).

[2] Abolghasemi, P., A. Mazaheri, M. Shah and L. Boloni, "Pay attention! - robustifying a deep visuomotor policy through task-focused visual attention", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)", (2019).

[3] Aeronautiques, C., A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson *et al.*, "Pddl| the planning domain definition language", (1998).

[4] Albrecht, S. V. and P. Stone, "Autonomous agents modelling other agents: A comprehensive survey and open problems", Artif. Intell. **258**, 66–95, URL `https://doi.org/10.1016/j.artint.2018.01.002` (2018).

[5] Alzaidy, R., C. Caragea and C. L. Giles, "Bi-lstm-crf sequence labeling for keyphrase extraction from scholarly documents", in "The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019", pp. 2551–2557 (2019).

[6] Amir, O. and Y. K. Gal, "Plan recognition in virtual laboratories", in "Proceedings of IJCAI", pp. 2392–2397 (2011).

[7] Anderson, P., X. He, C. Buehler, D. Teney, M. Johnson, S. Gould and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering", in "CVPR", vol. 3, p. 6 (2018).

[8] Anderson, S. J., N. Yamagishi and V. Karavia, "Attentional processes link perception and action", Proceedings of the Royal Society of London. Series B: Biological Sciences **269**, 1497, 1225–1232 (2002).

[9] Avrahami-Zilberbrand, D. and G. A. Kaminka, "Keyhole adversarial plan recognition for recognition of suspicious and anomalous behavior", (2014).

[10] Banerjee, S., A. Hati, S. Chaudhuri and R. Velmurugan, "Cosegnet: Image cosegmentation using a conditional siamese convolutional network.", in "IJCAI", pp. 673–679 (2019).

[11] Bensch, S. and T. Hellström, "On ambiguity in robot learning from demonstration", in "Intelligent autonomous systems", pp. 47–56 (Citeseer, 2010).

[12] Branavan, S., N. Kushman, T. Lei and R. Barzilay, "Learning high-level planning from text", in "Proceedings of ACL-12", (2012).

[13] Branicky, M. S., V. S. Borkar and S. K. Mitter, "A unified framework for hybrid control: Model and optimal control theory", IEEE transactions on automatic control **43**, 1, 31–45 (1998).

[14] Breazeal, C., M. Berlin, A. Brooks, J. Gray and A. L. Thomaz, "Using perspective taking to learn from ambiguous demonstrations", Robotics and autonomous systems **54**, 5, 385–393 (2006).

[15] Bromley, J., I. Guyon, Y. LeCun, E. Säckinger and R. Shah, "Signature verification using a" siamese" time delay neural network", in "Advances in neural information processing systems", pp. 737–744 (1994).

[16] Brown, D. S., S. Niekum and M. Petrik, "Bayesian robust optimization for imitation learning", arXiv preprint arXiv:2007.12315 (2020).

[17] Bryce, D., D. Whitebread and D. Szűcs, "The relationships among executive functions, metacognitive skills and educational achievement in 5 and 7 year-old children", Metacognition and Learning **10**, 2, 181–198 (2015).

[18] Bui, H. H., "A general model for online probabilistic plan recognition", in "Proceedings of IJCAI", pp. 1309–1318 (2003).

[19] Bui, H. H., S. Venkatesh and G. A. W. West, "Policy recognition in the abstract hidden markov model", J. Artif. Intell. Res. (JAIR) **17**, 451–499 (2002).

[20] Bulling, A., U. Blanke and B. Schiele, "A tutorial on human activity recognition using body-worn inertial sensors", ACM Comput. Surv. **46**, 3, 33 (2014).

[21] Camacho, A., R. T. Icarte, T. Q. Klassen, R. A. Valenzano and S. A. McIlraith, "Ltl and beyond: Formal languages for reward function specification in reinforcement learning.", in "IJCAI", vol. 19, pp. 6065–6073 (2019).

[22] Carreira, J., R. Caseiro, J. Batista and C. Sminchisescu, "Semantic segmentation with second-order pooling", in "European Conference on Computer Vision", pp. 430–443 (Springer, 2012).

[23] Chen, K., L. Yao, D. Zhang, B. Guo and Z. Yu, "Multi-agent attentional activity recognition", CoRR **abs/1905.08948**, URL `http://arxiv.org/abs/1905.08948` (2019).

[24] Chen, L.-C., G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs", arXiv preprint arXiv:1412.7062 (2014).

[25] Chu, Y., Y. C. Song, R. Levinson and H. A. Kautz, "Interactive activity recognition and prompting to assist people with cognitive disabilities", JAISE **4**, 5, 443–459 (2012).

[26] Coumans, E. and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning", (2016).

[27] Cox, M. T. and A. Ram, "Introspective multistrategy learning: On the construction of learning strategies", Tech. rep., WRIGHT STATE UNIV DAYTON OH DEPT OF COMPUTER SCIENCE AND ENGINEERING (1999).

[28] Cruz Jr, G. V., Y. Du and M. E. Taylor, "Pre-training neural networks with human demonstrations for deep reinforcement learning", arXiv preprint arXiv:1709.04083 (2017).

[29] de Granville, C., J. Southerland and A. H. Fagg, "Learning grasp affordances through human demonstration", in "Proceedings of the International Conference on Development and Learning (ICDL'06)", (2006).

[30] de Haan, P., D. Jayaraman and S. Levine, "Causal confusion in imitation learning", in "Advances in Neural Information Processing Systems", pp. 11698–11709 (2019).

[31] Detry, R., C. H. Ek, M. Madry and D. Kragic, "Learning a dictionary of prototypical grasp-predicting parts from grasping experience", in "2013 IEEE International Conference on Robotics and Automation", pp. 601–608 (IEEE, 2013).

[32] Dong, X., A. Y. Halevy, J. Madhavan, E. Nemes and J. Zhang, "Simlarity search for web services", in "Proceedings of VLDB", pp. 372–383 (2004).

[33] E-Martín, Y., M. D. R.-Moreno and D. E. Smith, "A fast goal recognition technique based on interaction estimates", in "Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015", pp. 761–768 (2015), URL `http://ijcai.org/Abstract/15/113`.

[34] Fikes, R. E. and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving", Artificial intelligence **2**, 3-4, 189–208 (1971).

[35] Finn, C., P. Christiano, P. Abbeel and S. Levine, "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models", arXiv preprint arXiv:1611.03852 (2016).

[36] Finn, C., S. Levine and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization", in "International conference on machine learning", pp. 49–58 (2016).

[37] Freedman, R. G., H.-T. Jung and S. Zilberstein, "Plan and activity recognition from a topic modeling perspective", in "Proceedings of ICAPS", (2014).

[38] Fu, J., K. Luo and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning", arXiv preprint arXiv:1710.11248 (2017).

[39] Fujimoto, S., H. Van Hoof and D. Meger, "Addressing function approximation error in actor-critic methods", arXiv preprint arXiv:1802.09477 (2018).

[40] Fukunaga, K. and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors", IEEE transactions on computers **100**, 7, 750–753 (1975).

[41] Gao, Y., H. Xu, J. Lin, F. Yu, S. Levine and T. Darrell, "Reinforcement learning from imperfect demonstrations", arXiv preprint arXiv:1802.05313 (2018).

[42] Geib, C. W. and R. P. Goldman, "A probabilistic plan recognition algorithm based on plan tree grammars", Artificial Intelligence **173**, 11, 1101–1132 (2009).

[43] Ghasemi, M., E. Bulgur and U. Topcu, "Task-oriented active perception and planning in environments with partially known semantics", in "International Conference on Machine Learning", pp. 3484–3493 (PMLR, 2020).

[44] Gibson, J. J., *The ecological approach to visual perception: classic edition* (Psychology Press, 2014).

[45] Girdhar, R. and D. Ramanan, "Attentional pooling for action recognition", in "Advances in Neural Information Processing Systems", pp. 34–45 (2017).

[46] Graves, A., "Generating sequences with recurrent neural networks", CoRR **abs/1308.0850**, URL `http://arxiv.org/abs/1308.0850` (2013).

[47] Graves, A., A. Mohamed and G. E. Hinton, "Speech recognition with deep recurrent neural networks", in "IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013", pp. 6645–6649 (2013), URL `https://doi.org/10.1109/ICASSP.2013.6638947`.

[48] Haarnoja, T., H. Tang, P. Abbeel and S. Levine, "Reinforcement learning with deep energy-based policies", in "ICML'17 Proceedings of the 34th International Conference on Machine Learning - Volume 70", pp. 1352–1361 (2017).

[49] Haarnoja, T., A. Zhou, P. Abbeel and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", arXiv preprint arXiv:1801.01290 (2018).

[50] Hester, T., M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold *et al.*, "Deep q-learning from demonstrations", arXiv preprint arXiv:1704.03732 (2017).

[51] Hodges, M. R. and M. E. Pollack, "An 'object-use fingerprint': The use of electronic sensors for human identification", in "Ubicomp", pp. 289–303 (2007).

[52] Hoffmann, J., J. Porteous and L. Sebastia, "Ordered landmarks in planning", J. Artif. Intell. Res. **22**, 215–278, URL `https://doi.org/10.1613/jair.1492` (2004).

[53] Holtzen, S., Y. Zhao, T. Gao, J. B. Tenenbaum and S.-C. Zhu, "Inferring human intent from video by sampling hierarchical plans", in "Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on", pp. 1489–1496 (IEEE, 2016).

[54] Hsiao, K. and T. Lozano-Perez, "Imitation learning of whole-body grasps", in "2006 IEEE/RSJ International Conference on Intelligent Robots and Systems", pp. 5657–5662 (2006).

[55] Huang, T.-W., Y.-A. Wei, H.-T. Chen and J. Liu, "Object discovery in depth images", in "2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)", pp. 1–5 (IEEE, 2016).

[56] Kabanza, F., J. Filion, A. R. Benaskeur and H. Irandoust, "Controlling the hypothesis space in probabilistic plan recognition", in "IJCAI", (2013).

[57] Kambhampati, S., "Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain theories", in "Proceedings of AAAI", pp. 1601–1605 (2007).

[58] Kambhampati, S., S. Sreedharan, M. Verma, Y. Zha and L. Guan, "Symbols as a lingua franca for bridging human-ai chasm for explainable and advisable ai systems", arXiv preprint arXiv:2109.09904 (2021).

[59] Kautz, H. A. and J. F. Allen, "Generalized plan recognition", in "Proceedings of AAAI", pp. 32–37 (1986).

[60] Kıcıman, E. and M. Richardson, "Towards decision support and goal achievement: Identifying action-outcome relationships from social media", in "Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", pp. 547–556 (ACM, 2015).

[61] Kitani, K. M., B. D. Ziebart, J. A. Bagnell and M. Hebert, "Activity forecasting", in "European Conference on Computer Vision", pp. 201–214 (Springer, 2012).

[62] Kleeberger, K., R. Bormann, W. Kraus and M. F. Huber, "A survey on learning-based robotic grasping", Current Robotics Reports pp. 1–11 (2020).

[63] Koch, G., R. Zemel and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition", in "ICML deep learning workshop", vol. 2 (Lille, 2015).

[64] Körding, K. P., U. Beierholm, W. J. Ma, S. Quartz, J. B. Tenenbaum and L. Shams, "Causal inference in multisensory perception", PLoS one **2**, 9, e943 (2007).

[65] Kosiorek, A., A. Bewley and I. Posner, "Hierarchical attentive recurrent tracking", in "Advances in Neural Information Processing Systems", pp. 3053–3061 (2017).

[66] Kostov, K. and A. Janyan, "The role of attention in the affordance effect: can we afford to ignore it?", Cognitive processing **13**, 1, 215–218 (2012).

[67] Kulkarni, A., Y. Zha, T. Chakraborti, S. G. Vadlamudi, Y. Zhang and S. Kambhampati, "Explicablility as minimizing distance from expected behavior", arXiv preprint arXiv:1611.05497 (2016).

[68] Kulkarni, A., Y. Zha, T. Chakraborti, S. G. Vadlamudi, Y. Zhang and S. Kambhampati, "Explicable planning as minimizing distance from expected behavior", in "AAMAS Conference proceedings", (2019).

[69] Lake, B. M., R. Salakhutdinov and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction", Science **350**, 6266, 1332–1338 (2015).

[70] LeCun, Y., Y. Bengio and G. E. Hinton, "Deep learning", Nature **521**, 7553, 436–444, URL `https://doi.org/10.1038/nature14539` (2015).

[71] Lesh, N. and O. Etzioni, "A sound and fast goal recognizer", in "IJCAI", pp. 1704–1710 (1995).

[72] Lester, J., T. Choudhury, N. Kern, G. Borriello and B. Hannaford, "A hybrid discriminative/generative approach for modeling human activities", in "IJCAI", pp. 766–772 (2005).

[73] Levine, S. J. and B. C. Williams, "Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams", J. Artif. Intell. Res. **63**, 281–359, URL `https://doi.org/10.1613/jair.1.11243` (2018).

[74] Li, S., S. Song, G. Huang, Z. Ding and C. Wu, "Domain invariant and class discriminative feature learning for visual domain adaptation", IEEE Transactions on Image Processing **27**, 9, 4260–4273 (2018).

[75] Liao, L., D. J. Patterson, D. Fox and H. A. Kautz, "Learning and inferring transportation routines", Artif. Intell. **171**, 5-6, 311–331 (2007).

[76] Lipowski, A. and D. Lipowska, "Roulette-wheel selection via stochastic acceptance", Physica A: Statistical Mechanics and its Applications **391**, 6, 2193–2196 (2012).

[77] Long, J., E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 3431–3440 (2015).

[78] Lopes, M., F. S. Melo and L. Montesano, "Affordance-based imitation learning in robots", in "2007 IEEE/RSJ international conference on intelligent robots and systems", pp. 1015–1021 (IEEE, 2007).

[79] Lu, X., W. Wang, C. Ma, J. Shen, L. Shao and F. Porikli, "See more, know more: Unsupervised video object segmentation with co-attention siamese networks", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 3623–3632 (2019).

[80] Lyons, K. E. and P. D. Zelazo, "Monitoring, metacognition, and executive function: Elucidating the role of self-reflection in the development of self-regulation", in "Advances in child development and behavior", vol. 40, pp. 379–412 (Elsevier, 2011).

[81] Lyu, D., F. Yang, B. Liu and S. Gustafson, "Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning", in "Proceedings of the AAAI Conference on Artificial Intelligence", vol. 33, pp. 2970–2977 (2019).

[82] Mahmud, T., M. Hasan and A. K. Roy-Chowdhury, "Joint prediction of activity labels and starting times in untrimmed videos", in "Computer Vision (ICCV), 2017 IEEE International Conference on", pp. 5784–5793 (IEEE, 2017).

[83] Mandikal, P. and K. Grauman, "Dexterous robotic grasping with object-centric visual affordances", (2020).

[84] Massardi, J., M. Gravel and E. Beaudry, "Error-tolerant anytime approach to plan recognition using a particle filter", in "Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019.", pp. 284–291 (2019), URL `https://aaai.org/ojs/index.php/ICAPS/article/view/3490`.

[85] May, S., M. Klodt, E. Rome and R. Breithaupt, "Gpu-accelerated affordance cueing based on visual attention", in "2007 IEEE/RSJ International Conference on Intelligent Robots and Systems", pp. 3385–3390 (IEEE, 2007).

[86] Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Distributed representations of words and phrases and their compositionality", in "NIPS", pp. 3111–3119 (2013).

[87] Minka, T. *et al.*, "Divergence measures and message passing", Tech. rep., Technical report, Microsoft Research (2005).

[88] Mirsky, R. and Y. K. Gal, "SLIM: semi-lazy inference mechanism for plan recognition", in "Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016", pp. 394–400 (2016), URL `http://www.ijcai.org/Abstract/16/063`.

[89] Morales, O. C. F. and R. F. De la Rosa, "Ambiguity analysis in learning from demonstration applications for mobile robots", in "2013 16th International Conference on Advanced Robotics (ICAR)", pp. 1–6 (IEEE, 2013).

[90] Morton, J. and M. J. Kochenderfer, "Simultaneous policy learning and latent state inference for imitating driver behavior", in "2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)", pp. 1–6 (IEEE, 2017).

[91] Mostajabi, M., P. Yadollahpour and G. Shakhnarovich, "Feedforward semantic segmentation with zoom-out features", in "The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", (2015).

[92] Mukherjee, P., B. Lall and S. Lattupally, "Object cosegmentation using deep siamese network", arXiv preprint arXiv:1803.02555 (2018).

[93] Nair, A., B. McGrew, M. Andrychowicz, W. Zaremba and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations", in "2018 IEEE International Conference on Robotics and Automation (ICRA)", pp. 6292–6299 (IEEE, 2018).

[94] Navalpakkam, V. and L. Itti, "An integrated model of top-down and bottom-up attention for optimizing detection speed", in "null", pp. 2049–2056 (ieee, 2006).

[95] Nelson, T. O., "Metamemory: A theoretical framework and new findings", in "Psychology of learning and motivation", vol. 26, pp. 125–173 (Elsevier, 1990).

[96] Nelson, T. O. and L. Narens, "Why investigate metacognition", Metacognition: Knowing about knowing **13**, 1–25 (1994).

[97] Nepovinnykh, E., T. Eerola and H. Kalviainen, "Siamese network based pelage pattern matching for ringed seal re-identification", in "Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops", (2020).

[98] Ng, A. Y., D. Harada and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping", in "ICML", vol. 99, pp. 278–287 (1999).

[99] Ng, A. Y., H. J. Kim, M. I. Jordan and S. Sastry, "Autonomous helicopter flight via reinforcement learning", in "Proceedings of NIPS-03", (2003).

[100] Noh, H., S. Hong and B. Han, "Learning deconvolution network for semantic segmentation", in "Proceedings of the IEEE international conference on computer vision", pp. 1520–1528 (2015).

[101] Oaksford, M., N. Chater *et al.*, *Bayesian rationality: The probabilistic approach to human reasoning* (Oxford University Press, 2007).

[102] Oh, S., A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis *et al.*, "A large-scale benchmark dataset for event recognition in surveillance video", in "CVPR 2011", pp. 3153–3160 (IEEE, 2011).

[103] Pan, J. J., Q. Yang and S. J. Pan, "Online co-localization in indoor wireless networks by dimension reduction", in "AAAI", pp. 1102–1107 (2007).

[104] Pan, S. J., J. T. Kwok, Q. Yang and J. J. Pan, "Adaptive localization in a dynamic wifi environment through multi-view learning", in "AAAI", pp. 1108–1113 (2007).

[105] Pascanu, R., Ç. Gülçehre, K. Cho and Y. Bengio, "How to construct deep recurrent neural networks", in "2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings", (2014), URL `http://arxiv.org/abs/1312.6026`.

[106] Pereira, R. F., N. Oren and F. Meneguzzi, "Landmark-based heuristics for goal recognition", in "Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.", pp. 3622–3628 (2017), URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14666.

[107] Pereira, R. F., A. G. Pereira and F. Meneguzzi, "Landmark-enhanced heuristics for goal recognition in incomplete domain models", in "Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019.", pp. 329–337 (2019), URL https://aaai.org/ojs/index.php/ICAPS/article/view/3495.

[108] Pfeiffer, M., S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations", IEEE Robotics and Automation Letters **3**, 4, 4423–4430 (2018).

[109] Pollack, M. E., L. E. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan and I. Tsamardinos, "Autominder: an intelligent cognitive orthotic system for people with memory impairment", Robotics and Autonomous Systems **44**, 3-4, 273–282 (2003).

[110] Qian, H., S. J. Pan and C. Miao, "Distribution-based semi-supervised learning for activity recognition", in "The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.", pp. 7699–7706 (2019), URL https://aaai.org/ojs/index.php/AAAI/article/view/4765.

[111] Qian, K., X. Jing, Y. Duan, B. Zhou, F. Fang, J. Xia and X. Ma, "Grasp pose detection with affordance-based task constraint learning in single-view point clouds", Journal of Intelligent & Robotic Systems **100**, 145–163 (2020).

[112] Rajeswaran, A., V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations", arXiv preprint arXiv:1709.10087 (2017).

[113] Ramachandruni, K., M. Babu V., A. Majumder, S. Dutta and S. Kumar, "Attentive task-net: Self supervised task-attention network for imitation learning using video demonstration", in "2020 IEEE International Conference on Robotics and Automation (ICRA)", pp. 4760–4766 (2020).

[114] Ramırez, M. and H. Geffner, "Plan recognition as planning", in "Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc", pp. 1778–1783 (2009).

[115] Ramirez, M. and H. Geffner, "Plan recognition as planning", in "Proceedings of IJCAI", pp. 1778–1783 (2009).

[116] Ramírez, M. and H. Geffner, "Plan recognition as planning", in "IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009", pp. 1778–1783 (2009).

[117] Ramırez, M. and H. Geffner, "Probabilistic plan recognition using off-the-shelf classical planners", in "Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)", pp. 1121–1126 (Citeseer, 2010).

[118] Ramirez, M. and H. Geffner, "Probabilistic plan recognition using off-the-shelf classical planners", in "Proceedings of AAAI", pp. 1121–1126 (2010).

[119] Ren, A. Z., S. Veer and A. Majumdar, "Generalization guarantees for multi-modal imitation learning", arXiv preprint arXiv:2008.01913 (2020).

[120] Riggio, L., C. Iani, E. Gherri, F. Benatti, S. Rubichi and R. Nicoletti, "The role of attention in the occurrence of the affordance effect", Acta psychologica **127**, 2, 449–458 (2008).

[121] Roebers, C. M., "Executive function and metacognition: Towards a unifying framework of cognitive self-regulation", Developmental Review **45**, 31–51 (2017).

[122] Roebers, C. M. and E. Feurer, "Linking executive functions and procedural metacognition", Child Development Perspectives **10**, 1, 39–44 (2016).

[123] Rong, X., "word2vec parameter learning explained", arXiv preprint arXiv:1411.2738 (2014).

[124] Ross, S. and D. Bagnell, "Efficient reductions for imitation learning", in "Proceedings of the thirteenth international conference on artificial intelligence and statistics", pp. 661–668 (2010).

[125] Ross, S., G. Gordon and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning", in "Proceedings of the fourteenth international conference on artificial intelligence and statistics", pp. 627–635 (2011).

[126] Russell, S. and P. Norvig, "Artificial intelligence: a modern approach", (2002).

[127] Salimans, T. and R. Chen, "Learning montezuma's revenge from a single demonstration", arXiv preprint arXiv:1812.03381 (2018).

[128] Saria, S. and S. Mahadevan, "Probabilistic plan recognitionin multiagent systems", in "Proceedings of AAAI", (2004).

[129] Schroff, F., D. Kalenichenko and J. Philbin, "Facenet: A unified embedding for face recognition and clustering", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 815–823 (2015).

[130] Shi, X., Z. Chen, H. Wang, D. Yeung, W. Wong and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting", in "Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada", pp. 802–810 (2015).

[131] Siegler, R. S. *et al.*, "Microgenetic studies of self-explanation", Microdevelopment: Transition processes in development and learning pp. 31–58 (2002).

[132] Singla, P. and R. Mooney, "Abductive markov logic for plan recognition", in "Proceedings of AAAI", pp. 1069–1075 (2011).

[133] Sohrabi, S., A. V. Riabov and O. Udrea, "Plan recognition as planning revisited.", in "IJCAI", pp. 3258–3264 (2016).

[134] Sohrabi, S., A. V. Riabov and O. Udrea, "Plan recognition as planning revisited", in "Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016", pp. 3258–3264 (2016), URL `http://www.ijcai.org/Abstract/16/461`.

[135] Song, L., D. Gong, Z. Li, C. Liu and W. Liu, "Occlusion robust face recognition based on mask learning with pairwise differential siamese network", in "Proceedings of the IEEE International Conference on Computer Vision", pp. 773–782 (2019).

[136] Spaan, M. T., "Cooperative active perception using pomdps", in "AAAI 2008 workshop on advancements in POMDP solvers", (2008).

[137] Sreedharan, S., U. Soni, M. Verma, S. Srivastava and S. Kambhampati, "Bridging the gap: Providing post-hoc symbolic explanations for sequential decision-making problems with black box simulators", arXiv preprint arXiv:2002.01080 (2020).

[138] Srivastava, S., G. Sharma and B. Lall, "Large scale novel object discovery in 3d", in "2018 IEEE Winter Conference on Applications of Computer Vision (WACV)", pp. 179–188 (IEEE, 2018).

[139] Steil, J., F. Röthling, R. Haschke and H. Ritter, "Situated robot learning for multi-modal instruction and imitation of grasping", Robotics and Autonomous Systems **47**, 2, 129–141, URL `https://www.sciencedirect.com/science/article/pii/S0921889004000430`, robot Learning from Demonstration (2004).

[140] Sternberg, R., K. Sternberg and J. Mio, *Cognitive Psychology* (Wadsworth/Cengage Learning, 2011), URL `https://books.google.com/books?id=DIg5XwAACAAJ`.

[141] Sternberg, R. J., "A triarchic theory of human intelligence", in "Human assessment: Cognition and motivation", pp. 43–44 (Springer, 1986).

[142] Sternberg, R. J. and K. Sternberg, *Cognitive psychology* (Nelson Education, 2016).

[143] Sutton, R. S., D. Precup and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning", Artificial intelligence **112**, 1-2, 181–211 (1999).

[144] Sweeney, J. D. and R. Grupen, "A model of shared grasp affordances from demonstration", in "2007 7th IEEE-RAS International Conference on Humanoid Robots", pp. 27–35 (IEEE, 2007).

[145] Tian, X., H. H. Zhuo and S. Kambhampati, "Discovering underlying plans based on distributed representations of actions", arXiv preprint arXiv:1511.05662 (2015).

[146] Tian, X., H. H. Zhuo and S. Kambhampati, "Discovering underlying plans based on distributed representations of actions", in "Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems", pp. 1135–1143 (International Foundation for Autonomous Agents and Multiagent Systems, 2016).

[147] Toro Icarte, R., T. Q. Klassen, R. Valenzano and S. A. McIlraith, "Teaching multiple tasks to an rl agent using ltl", in "Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems", pp. 452–461 (2018).

[148] Tuan Nguyen, M. D., Subbarao Kambhampati, "Synthesizing robust plans under incomplete domain models", in "Proc. AAAI Workshop on Generalized Planning", (2011).

[149] Vecerik, M., T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards", arXiv preprint arXiv:1707.08817 (2017).

[150] Wang, X. and Q. Ji, "Video event recognition with deep hierarchical context model", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 4418–4427 (2015).

[151] Wen, Y., K. Zhang, Z. Li and Y. Qiao, "A discriminative feature learning approach for deep face recognition", in "European conference on computer vision", pp. 499–515 (Springer, 2016).

[152] Wise, M., M. Ferguson, D. King, E. Diehr and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications", in "Workshop on autonomous mobile service robots", (2016).

[153] Xie, C., C. Li, B. Zhang, C. Chen, J. Han and J. Liu, "Memory attention networks for skeleton-based action recognition", in "Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018,

July 13-19, 2018, Stockholm, Sweden.", pp. 1639–1645 (2018), URL https://doi.org/10.24963/ijcai.2018/227.

[154] Xie, D., T. Shu, S. Todorovic and S.-C. Zhu, "Learning and inferring "dark matter" and predicting human intents and trajectories in videos", IEEE transactions on pattern analysis and machine intelligence **40**, 7, 1639–1652 (2018).

[155] Yan, H., J. Lu and X. Zhou, "Prototype-based discriminative feature learning for kinship verification", IEEE Transactions on cybernetics **45**, 11, 2535–2545 (2014).

[156] Yang, F., D. Lyu, B. Liu and S. Gustafson, "Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making", arXiv preprint arXiv:1804.07779 (2018).

[157] Yang, J., J. Lu, S. Lee, D. Batra and D. Parikh, "Graph r-cnn for scene graph generation", in "Proceedings of the European Conference on Computer Vision (ECCV)", (2018).

[158] Yang, Q., K. Wu and Y. Jiang, "Learning action models from plan examples using weighted MAX-SAT", Artificial Intelligence Journal **171**, 107–143 (2007).

[159] Yen-Chen, L., A. Zeng, S. Song, P. Isola and T.-Y. Lin, "Learning to see before learning to act: Visual pre-training for manipulation", in "IEEE International Conference on Robotics and Automation (ICRA)", (2020), URL https://yenchenlin.me/vision2action/.

[160] Yin, J., D. Shen, Q. Yang and Z.-N. Li, "Activity recognition through goal-based segmentation", in "AAAI", pp. 28–34 (2005).

[161] Yin, J., Q. Yang and L. M. Ni, "Adaptive temporal radio maps for indoor location estimation", in "PerCom", pp. 85–94 (2005).

[162] Yu, S., B. Krishnapuram, R. Rosales and R. B. Rao, "Active sensing", in "Artificial Intelligence and Statistics", pp. 639–646 (2009).

[163] Zeng, A., S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching", in "2018 IEEE international conference on robotics and automation (ICRA)", pp. 3750–3757 (IEEE, 2018).

[164] Zha, Y., S. Bhambri and L. Guan, "Contrastively learning visual attention as affordance cues from demonstrations for robotic grasping", arXiv preprint arXiv:2104.00878 (2021).

[165] Zha, Y., L. Guan and S. Kambhampati, "Learning from ambiguous demonstrations with self-explanation guided reinforcement learning", in "Submitted to ICRA 2022 and will be on arXiv e-prints soon", (2021), https://drive.google.com/file/d/179Ixw-Grb2ob–ZNpENOeuNMQLcz6b1A/view.

[166] Zha, Y., Y. Li, S. Gopalakrishnan, B. Li and S. Kambhampati, "Recognizing plans by learning embeddings from observed action distributions", in "Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems", pp. 2153–2155 (2018).

[167] Zha, Y., Y. Li, S. Gopalakrishnan, B. Li and S. Kambhampati, "Recognizing plans by learning embeddings from observed action distributions", in "Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems", pp. 2153–2155 (International Foundation for Autonomous Agents and Multiagent Systems, 2018).

[168] Zha, Y., Y. Li, T. Yu, S. Kambhampati and B. Li, "Plan-recognition-driven attention modeling for visual recognition", arXiv preprint arXiv:1812.00301 (2018).

[169] Zhang, X., F. Pala and B. Bhanu, "Attributes co-occurrence pattern mining for video-based person re-identification", in "2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)", pp. 1–6 (IEEE, 2017).

[170] Zhang, Y., S. Sreedharan and S. Kambhampati, "Capability models and their applications in planning", in "Proceedings of AAMAS", pp. 1151–1159 (2015).

[171] Zhang, Y., S. Sreedharan, A. Kulkarni, T. Chakraborti, H. H. Zhuo and S. Kambhampati, "Plan explicability and predictability for robot task planning", in "2017 IEEE international conference on robotics and automation (ICRA)", pp. 1313–1320 (IEEE, 2017).

[172] Zheng, V. W. and Q. Yang, "User-dependent aspect model for collaborative activity recognition", in "IJCAI", pp. 2085–2090 (2011).

[173] Zheng, Y., W.-K. Wong, X. Guan and S. Trost, "Physical activity recognition from accelerometer data using a multi-scale ensemble method", in "IAAI", (2013).

[174] Zhuo, H. H., T. A. Nguyen and S. Kambhampati, "Model-lite case-based planning", in "AAAI", (2013).

[175] Zhuo, H. H., Q. Yang, D. H. Hu and L. Li, "Learning complex action models with quantifiers and implications", Artificial Intelligence **174**, 18, 1540–1569 (2010).

[176] Zhuo, H. H., Y. Zha, S. Kambhampati and X. Tian, "Discovering underlying plans based on shallow models", ACM Transactions on Intelligent Systems and Technology (TIST) **11**, 2, 1–30 (2020).

[177] Ziebart, B. D., A. L. Maas, J. A. Bagnell and A. K. Dey, "Maximum entropy inverse reinforcement learning.", in "Aaai", vol. 8, pp. 1433–1438 (Chicago, IL, USA, 2008).