Neural Bayesian Goal Inference for Symbolic Planning Domains

by

Jordyn Mann

S.B., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
January 15, 2021

Certified by . . . .
Vikash Mansinghka
Principal Research Scientist
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Neural Bayesian Goal Inference for Symbolic Planning Domains

by

Jordyn Mann

Submitted to the Department of Electrical Engineering and Computer Science
on January 15, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

There are several reasons for which one may aim to infer the short- and long-term goals of agents in diverse physical domains. As increasingly powerful autonomous systems come into development, it is conceivable that they may eventually need to accurately infer the goals of humans. There are also more immediate reasons for which this sort of inference may be desirable, such as in the use case of intelligent personal assistants. This thesis introduces a neural Bayesian approach to goal inference in multiple symbolic planning domains and compares the results of this approach to the results of a recently developed Monte Carlo Bayesian inference method known as Sequential Inverse Plan Search (SIPS). SIPS is based on sequential Monte Carlo inference for Bayesian inversion of probabilistic plan search in Planning Domain Definition Language (PDDL) domains. In addition to the neural architectures, the thesis also introduces approaches for converting PDDL predicate state representations to numerical arrays and vectors suitable for input to the neural networks. The experimental results presented indicate that for the domains investigated, in cases where the training set is representative of the test set, the neural approach provides similar accuracy results to SIPS in the later portions of the observation sequences with a far shorter amortized time cost. However, in earlier timesteps of those observation sequences and in cases where the training set is less similar to the testing set, SIPS outperforms the neural approach in terms of accuracy. These results indicate that a model-based inference method where SIPS uses a neural proposal based on the neural networks designed in this thesis could have the potential to combine the advantages of both goal inference approaches by improving the speed of SIPS inference while maintaining generalizability and high accuracy throughout the timesteps of the observation sequences.

Thesis Supervisor: Vikash Mansinghka
Title: Principal Research Scientist

Acknowledgments

I would like to thank everyone in the lab who has helped me from two years ago when I began as a UROP until now, as I submit my thesis. In particular, I'd like to thank my thesis supervisor, Vikash Mansinghka, who has given me a plethora of advice and helped guide my research along the way. I'd also like to thank Tan Zhi-Xuan, who has become a mentor to me over the past year and has given me a massive amount of advice on my thesis. The discussions I've had with them have furthered my understanding of neural networks, probabilistic computing, and the field of computer science in general immensely.

I would of course also like to thank my friends and family, though there is nothing I can say that would do justice to the love and support they've given me over the years.

This research was supported by the DARPA Machine Common Sense program, under contract number CW3013540.

Contents

1 Introduction **13**

1.1 Related Work 15

2 Background on Bayesian Goal Inference **17**

2.1 Goal Prior 17

2.2 Generative Model for Replanning Agents 18

2.3 Posterior 22

3 Converting PDDL Programs to Vector Representations **27**

3.1 Technical Challenges 27

3.2 Conversion of Blocks World 28

3.3 Conversion of Doors-Keys-Gems 30

4 Neural Goal Inference via LSTMs **35**

4.1 Mathematical Formulation 35

4.2 Neural Architecture for Blocks World Domain 36

4.3 Neural Architecture for Doors-Keys-Gems Domain 37

4.4 Comparing Monte Carlo and Neural Implementations of Bayesian Goal Inference 39

5 Conclusion **47**

5.1 Contributions 48

5.2 Limitations and Future Work 48

List of Figures

1-1 How would you go about inferring goals of an agent? 14

2-1 Deciding whether replanning is needed. 19

2-2 A demonstration of the goal inference model predicting which gem the agent aims to obtain. 25

3-1 Conversion from a Blocks World PDDL state to a vector representation for input to the LSTM. 30

3-2 Conversion from a Doors-Keys-Gems PDDL state to a representation that entails an array and a vector for input to the LSTM. 32

4-1 Diagram of the architecture of the Blocks World neural network. . . . 37

4-2 Diagram of the architecture of the Doors-Keys-Gems neural network. 38

4-3 Violin plots of the raw true goal predicted posteriors in the Blocks World domain over the four quartiles. 42

4-4 Violin plots of the raw true goal predicted posteriors in the Doors-Keys-Gems domain over the four quartiles. 43

List of Tables

4.1 The accuracy results of the two approaches on the Blocks World and Doors-Keys-Gems domains. 41

4.2 Computation cost measurements for the experiments in the Blocks World and Doors-Keys-Gems domains. 44

Chapter 1

Introduction

There are multiple reasons for which one may aim to infer the short- and long-term goals of agents in diverse physical domains, including to figure out how to help the agent achieve their goals. Inference of agent goals based on the actions they take has been attempted using many methods [8, 7, 10, 14]. However, most of these methods perform poorly if the agent does not use an optimal planning strategy, or if the environment state space is large. This thesis intends to accompany other research occurring within MIT's Probabilistic Computing Project to infer agent goals online with replanning, such that the inference model can predict goal posteriors given suboptimal agent planning and a large environment state space in a human-like manner. This thesis explores the benefits and drawbacks a neural approach as opposed to the probabilistic programming approach, Sequential Inverse Plan Search (SIPS), used in the accompanying research [13].

SIPS uses a Bayesian goal inference approach, which consists of a generative model as a prior and an inference function that receives partial observations and outputs a posterior calculated via conditional probabilities. SIPS has been shown to make accurate predictions of goal posteriors given suboptimal agent planning and a moderate sized environment state space in a human-like manner. The goal of this thesis is to develop the infrastructure necessary to train a neural network with example action sequences generated using the SIPS generative model and compare the accuracy and computational resource requirements between SIPS and the neural approach across

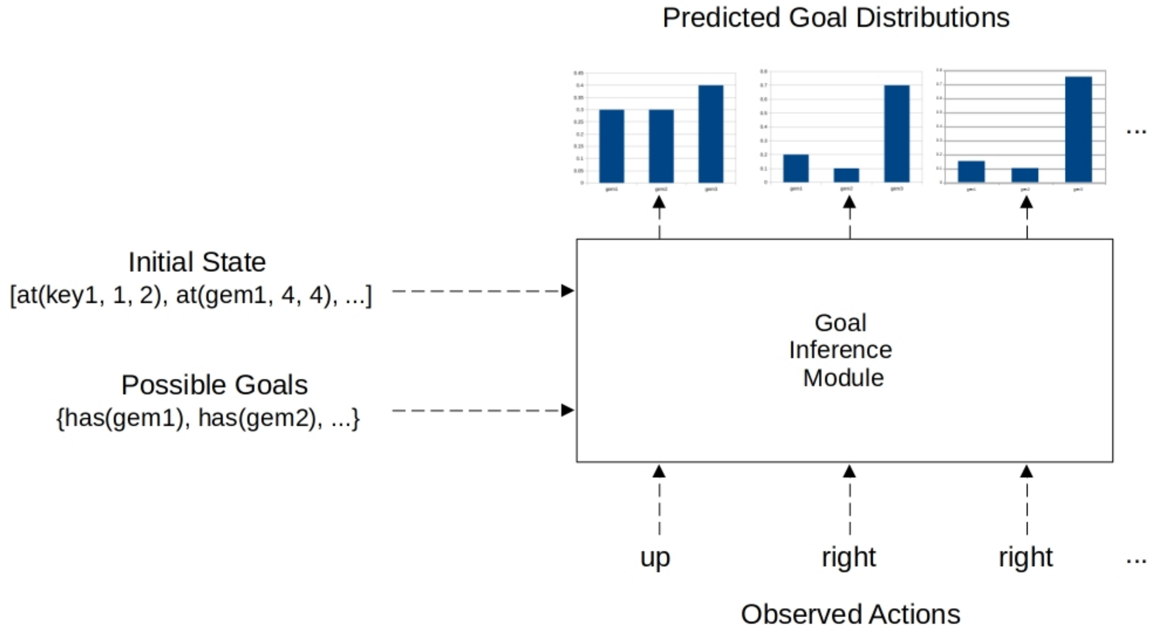

Figure 1-1: How would you go about inferring goals of an agent? Both modules discussed in this paper take as input a representation of the initial world state, a discrete set of possible goals, and a sequence of actions taken by the agent of interest. As it receives each observed action, the given inference module outputs a predicted posterior over the possible goals.

multiple planning domains and with worlds containing various numbers of objects.

Both the SIPS method and the neural approach analyzed in this thesis have a common goal inference structure, shown in Figure 1-1. Both approaches are initialized with an initial world state and a discrete set of possible goals. Then, they both sequentially take as input observed agent actions and, using their respective inference approaches, output for each observation a posterior over the possible goals given the observed actions up to that point. SIPS has been shown to accurately infer agent goals with reasonable amounts of computation in worlds with a moderate number of objects using a probabilistic programming approach. However, in a world with many objects and actions, using the SIPS methodology of inverting planning online may still be intractable. A neural approach, on the other hand, may take significant computation to train, but less computation at run time, since no complex planning inversions need occur. This would show that the neural approach provides a benefit in cases where one neural network can be trained and used on many inference problems of interest,

effectively amortizing the training cost over all of these inference executions. This thesis investigates whether this computation hypothesis seems to be empirically true and whether the results of the neural approach tend to be accurate and generalizable enough to consider it a viable approach to goal inference.

1.1 Related Work

In recent years, neural networks have been increasingly used for the purpose of approximate Bayesian inference over latent variables in generative models and probabilistic programs. One such approach is deep amortized variational inference, which is used to infer the parameters of a probabilistic model [11]. The researchers trained a neural network to map from an input y, like an observation sequence in terms of this thesis, to an approximate posterior $q(x|y)$. Then, the optimization step in the variational inference need only occur once for the learned function to be able to predict a posterior distribution for any input y. The approach in this thesis also takes advantage of amortization in regards to considering the computational resource requirements of the neural approach over a set of test observations. However, unlike the approach used in deep variational inference, where neural networks are trained to minimize the KL divergence between an approximate posterior and the true posterior, the neural networks implemented in this thesis are trained using a supervised learning approach. Given a synthetic dataset of observed agent trajectories and their corresponding goals, the networks are trained to map an observation sequence to a predicted distribution on goals at each timestep.

Neural networks have also been deployed specifically for the purpose of inferring the mental states of agents. One such approach, Machine Theory of Mind, uses meta-learning to model the mental state of agents it encounters [9]. Unlike the neural architectures presented in this thesis, those in the meta-learning approach learn to identify the unique behavior of many different agents and to predict the underlying mental states of the agents. Then, their approach uses these two learned approximations to predict subsequent actions of the agents. The networks presented in

this thesis, in contrast, predict the goal of an agent rather than its future actions. Furthermore, they do so directly from the observation sequence rather than by first predicting characteristics of the agent and its mental state.

Chapter 2

Background on Bayesian Goal Inference

The Sequential Inverse Plan Search (SIPS) method to which the neural approach discussed in this thesis will be compared is a Bayesian goal inference approach, consisting of a generative model and an inference procedure. Since the neural networks studied in this thesis are trained on data generated by the SIPS generative model, it is also considered a Bayesian goal inference approach. In the neural approach, rather than using an inference function that explicitly computes a posterior using conditional probabilities, the generative model is used to randomly sample agent trajectories and a neural network is trained to infer goals based on these samples. The various components of the generative model used in both approaches will be described in detail in the following sections, and pseudocode for the generative model can be found in Algorithm 1.

2.1 Goal Prior

The goal prior, or the expected probability distribution across possible agent goals before any observations are made, is easily customizable. Given the discrete set of goals that the agent may attempt to achieve, a simple prior may assume that any of these goals are equally likely. Thus, the goal in that case is sampled uniformly. The

integration of this function into the overall generative model is shown in a subsequent section. If some goals are initially considered more likely than others, this goal prior function can be easily be written to sample from the desired distribution, calling out to the Gen probabilistic programming library also developed within the Probabilistic Computing Project [2]. The rewritten function can then simply be used as the SAMPLE-GOAL-STATE function instead, and will be called properly in the overall generative model in Algorithm 1.

2.2 Generative Model for Replanning Agents

SIPS uses a Planning Domain Definition Language (PDDL) representation to describe the objects within the problem domain, the actions an agent can take, and the initial state of the world [6]. Plans, or action sequences, can be generated according to a planning algorithm of choice, where options include A* search and breadth-first search, among others, which work within the PDDL representation.

Plans generated by the goal inference generative model may involve replanning to simulate imperfect planning abilities in an agent and to limit computation. This replanning consists of creating a partial plan using a standard planning algorithm from the agent's current state toward a goal state, limited by some search budget. This replanning step may need to occur after any observed action in the sequence of observed actions, if the state the agent is in after an observed action is not one that was planned for in the previous partial plan. See Figure 2-1 for details on situations in which replanning is or is not required.

In order to simulate a realistic human agent, two parameters must be specified to describe the agent's replanning search budget distribution: p and r. p can be interpreted as the probability of continuing to search after expanding each search node, and r is the number of times the agent must consider giving up before actually ceasing to search. Then, a negative binomial distribution can be used to generate the maximum number of planning node expansions the agent is willing to make before ending the current stage of replanning and returning a partial plan. The following

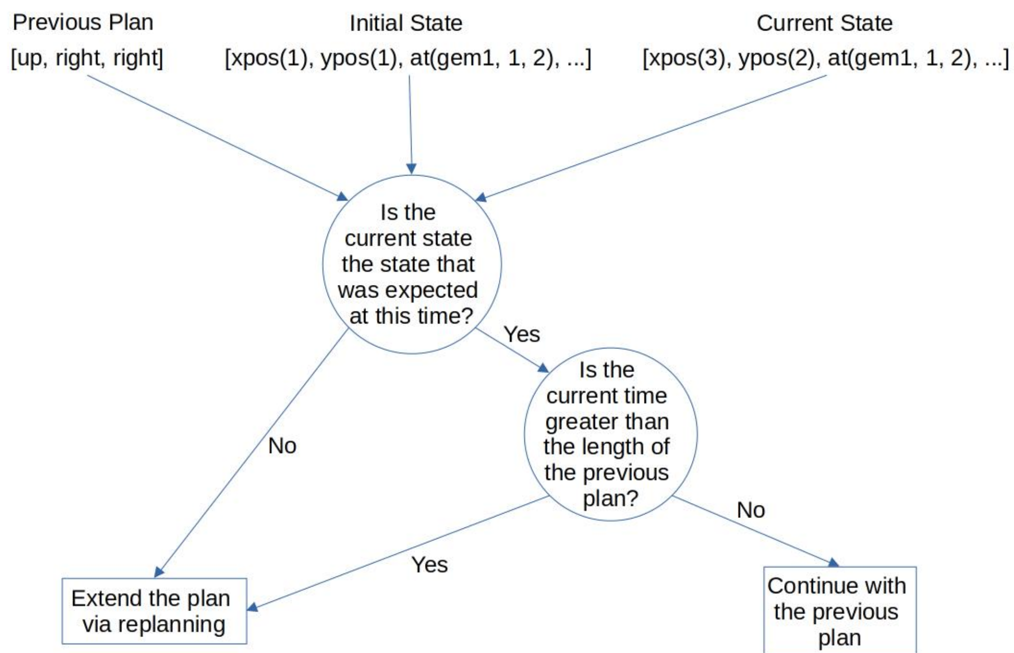

Figure 2-1: Deciding whether replanning is needed. The determination is made based on the previous plan and world state trajectory, which can be obtained using the initial world state. If the current world state was not anticipated in the previous plan, or if the number of steps already taken exceeds the number of steps planned for in the previous plan, replanning occurs. Otherwise, the algorithm continues with the previous plan.

equation describes this distribution, where k is the number of successes.

$$\Pr(X = k) = \binom{k + r - 1}{k} p^r (1 - p)^k \tag{2.1}$$

To further capture within the model the suboptimality of human planning, the A* plan generation search algorithm used within the SIPS experiments is probabilistic and does not necessarily always select the optimal node to expand at each step of the search execution. Rather, the node is selected with a probability proportional to its estimated cost, where less costly nodes are expanded with higher probability. Specifically, the probability P_{expand} of expanding a given node s in the queue is given by the following equation:

$$P_{expand}(s) \propto \exp(-f(s, g)/T). \tag{2.2}$$

where T is a temperature parameter that controls the randomness of search and $f(s, g)$ is given by:

$$f(s, g) = c(s) + h(s, g) \tag{2.3}$$

Noise may also be generated in the generative model to obscure observations. Such noise can be introduced by adding Gaussian noise to numeric state variables, or to boolean predicates by corrupting the predicate with a certain probability.

All of these sources of randomness come together to produce the generative WORLD-MODEL function and the algorithms to which it calls out, as shown in Algorithm 1. The model of the world is initialized in WORLD-MODEL, and the plan generation process unfolds through sequential applications of the WORLD-STEP function. The WORLD-STEP function in turn calls out to other generative functions that step forward the environment, observations, and plans individually, incorporating the sources of randomness described above.

Algorithm 1 Generative Pseudocode for Replanning Model

```
procedure WORLD-MODEL(r, replanner; d, domain; s0, initial state; op, observation parameters)
    parameters: n, number of steps
    g ← GET-GOALS(d)                                        ▷ Get the possible goals in the domain
    gs ← SAMPLE-GOAL-STATE(g)                               ▷ Initialize a sampled goal state
    rs ← (0, [], [s0], false) ▷ Initialize replan state relative step, partial plan and trajectory, and completion boolean
    es ← s0                                 ▷ Initialize the environment state to be the initial state
    os ← s0                                 ▷ Initialize the observation state to be the initial state
    wt ← UNFOLD(WORLD-STEP, n, gs, rs, es, os, op, r, d)    ▷ Sequentially apply WORLD-STEP
    return wt
end procedure

procedure WORLD-STEP(t, timestep; gs, goal state; rs, replan state; es, environment state; os, observation state;
op, observation parameters; r, replanner; d, domain)
    a ← GET-ACTION(rs)                                      ▷ Extract the action from the replan state
    es ← ENV-STEP(es, a, d)                                 ▷ Step forward the environment
    os ← OBS-STEP(es, op, d)                                ▷ Step forward the observations
    ps ← REPLAN-STEP(t, rs, r, d, es, gs)                   ▷ Step forward the replanning
    return gs, ps, es, os, r, d
end procedure

procedure ENV-STEP(es, environment state; a, action; d, domain)
    return EXECUTE-DETERMINISTIC-ACTION(d, es, a)
end procedure

procedure OBS-STEP(es, environment state; op, observation parameters; d, domain)
    os ← COPY(es)
    for (pt, (pd, pa)) ∈ op do        ▷ For each Prolog-style parameter term, distribution, and argument set...
        if HAS-NO-VARIABLES(pt) then                        ▷ If the term has no variables...
            ts ← [pt]          ▷ It is already a ground term and can be added to the terms list directly
        else if IS-FORALL(pt) then      ▷ If the term applies to all variables of a certain characteristic...
            c, b ← GET-TERM-ARGS(pt)                ▷ Extract term's application conditions and body
            fs ← SATISFY(c, es, d)                  ▷ Get facts that satisfy application conditions
            ts ← [SUBSTITUTE(b, f) for f ∈ fs]      ▷ Substitute those facts into the term's body
        else
            fs ← SATISFY(c, es, d)                  ▷ Get facts that satisfy the term
            ts ← [SUBSTITUTE(b, f) for f ∈ fs]      ▷ Substitute those facts in for the term
        end if
        for t ∈ ts do                                       ▷ For each term...
            os_t ← pd(es_t, pa)   ▷ Apply noise to term parameter observation according to distribution
        end for
    end for
    return os
end procedure

procedure REPLAN-STEP(t, timestep; rs, replan state; r, replanner; d, domain; es, environment state; gs, goal
state)
    parameters: n, number of replanning attempts; pcon, planner continuation probability
    pc ← IS-PLAN-DONE(r)                    ▷ Extract boolean indicating planning completion
    rels ← GET-REL-STEP(r) + 1
    pt ← GET-PARTIAL-TRAJECTORY(r)
    s ← pt_rels
    if pc or SATISFY(gs, s, d) then         ▷ If the plan is done or the state satisfies the goal...
        pp, pt ← [], [s, s]
        return 1, pp, pt, true              ▷ Return an empty plan and terminate the search
    else if rels < LENGTH(pt) then          ▷ If the planned trajectory has not been exceeded...
        pp ← GET-PARTIAL-PLAN(r)
        return rels, pp, pt, pc                             ▷ Return the existing partial plan
    else                        ▷ Otherwise, make a new partial plan given the current state
        mr ← NEGATIVE-BINOMIAL(n, 1 − pcon)                 ▷ Sample a planner resource bound
        p ← GET-PLANNER(r)                                  ▷ Extract the planner from the replanner
        p ← SET-MAX-RESOURCE(p, mr)
        pp, pt ← SAMPLE-PLAN(p, d, s, gs)       ▷ Sample plan using a planner-specific method
        if pp = nothing or PP = 0 then
            pd ← IS-GOAL-UNREACHABLE(pp)                    ▷ Terminate if the goal is unreachable...
            pp, pt ← [], [s, s]                             ▷ And return an empty plan
        end if
    end if
    return 1, pp, pt, pd                                    ▷ Return the new plan
end procedure
```

2.3 Posterior

Given a set of possible goals g and a sequence of state observations $o_{1:t}$, which may contain noise as is the case in the example generative model in Algorithm 1, the goal of the model is to infer the agent's goal by computing the posterior as follows.

In the generative model, we generate a plan given a state, a previous plan, and a goal; an action given a state and the plan; a subsequent state given a state and the action; and a subsequent observation given the subsequent state. Then, given the Markov property of the model, we know that:

$$\begin{aligned}
P(o_{1:T}, s_{1:T}, a_{1:T-1}, p_{1:T-1}|g) =& P(o_2, s_2, a_1, p_1|o_1, s_1, g) && (2.4)\\
&\cdot P(o_1, s_1|g) && (2.5)\\
&\cdot \prod_{t=3}^{T} P(o_t, s_t, a_{t-1}, p_{t-1}|o_{t-1}, s_{t-1}, a_{t-2}, p_{t-2}, g) && (2.6)
\end{aligned}$$

where the p, s, and a variables are random latent variables representing plans, states, and actions respectively. Applying the chain rule to the term in 2.6 repeatedly, we find:

$$\begin{aligned}
P(o_t, s_t, a_{t-1}, p_{t-1}|o_{t-1}, s_{t-1}, a_{t-2}, p_{t-2}, g) = & && (2.7)\\
& P(o_t|s_t, a_{t-1}, p_{t-1}, o_{t-1}, s_{t-1}, a_{t-2}, p_{t-2}, g) && (2.8)\\
& \cdot P(s_t|a_{t-1}, p_{t-1}, o_{t-1}, s_{t-1}, a_{t-2}, p_{t-2}, g) && (2.9)\\
& \cdot P(a_{t-1}|p_{t-1}, o_{t-1}, s_{t-1}, a_{t-2}, p_{t-2}, g) && (2.10)\\
& \cdot P(p_{t-1}|o_{t-1}, s_{t-1}, a_{t-2}, p_{t-2}, g) && (2.11)
\end{aligned}$$

Applying our independence assumptions, we find:

$$P(o_t, s_t, a_{t-1}, p_{t-1}|o_{t-1}, s_{t-1}, a_{t-2}, p_{t-2}, g) = P(o_t|s_t) \tag{2.12}$$
$$\cdot P(s_t|a_{t-1}, s_{t-1}) \tag{2.13}$$
$$\cdot P(a_{t-1}|p_{t-1}, s_{t-1}) \tag{2.14}$$
$$\cdot P(p_{t-1}|s_{t-1}, p_{t-2}, g) \tag{2.15}$$

Applying chain rule to the first term on the right hand side of the Markov property equation in line 2.4, we find:

$$P(o_2, s_2, a_1, p_1|o_1, s_1, g) = P(o_2|s_2, a_1, p_1, o_1, s_1, g) \tag{2.16}$$
$$\cdot P(s_2|a_1, p_1, o_1, s_1, g) \tag{2.17}$$
$$\cdot P(a_1|p_1, o_1, s_1, g) \tag{2.18}$$
$$\cdot P(p_1|o_1, s_1, g) \tag{2.19}$$

Also by our independence assumptions, we can simplify this to:

$$P(o_2, s_2, a_1, p_1|o_1, s_1, g) = P(o_2|s_2) \cdot P(s_2|a_1, s_1) \cdot P(a_1|p_1, s_1) \cdot P(p_1|s_1, g) \tag{2.20}$$

Applying chain rule to the term in 2.5, we find:

$$P(o_1, s_1|g) = P(o_1|s_1, g) \cdot P(s_1|g) \tag{2.21}$$

Applying our independence assumptions, we can simplify this to:

$$P(o_1, s_1|g) = P(o_1|s_1) \cdot P(s_1) \tag{2.22}$$

Substituting these simplified terms back in to our Markov property equation (2.4-

2.6), we find:

$$
\begin{aligned}
P(o_{1:T}, & s_{1:T}, a_{1:T-1}, p_{1:T-1}|g) = \\
& P(o_2|s_2) \cdot P(s_2|a_1, s_1) \cdot P(a_1|p_1, s_1) \cdot P(p_1|s_1, g) \cdot P(o_1|s_1) \cdot P(s_1) \\
& \cdot \prod_{t=3}^{T} P(o_t|s_t) \cdot P(s_t|a_{t-1}, s_{t-1}) \cdot P(a_{t-1}|p_{t-1}, s_{t-1}) \cdot P(p_{t-1}|s_{t-1}, p_{t-2}, g)
\end{aligned}
\tag{2.23}
$$

Finally, we marginalize over $s_{1:T}, a_{1:T-1}$, and $p_{1:T-1}$, and find:

$$
\begin{aligned}
P(o_{1:T}|g) = & \\
& \sum_{\substack{s_{1:t} \\ a_{1:t} \\ p_{1:t}}} (P(o_2|s_2) \cdot P(s_2|a_1, s_1) \cdot P(a_1|p_1, s_1) \cdot P(p_1|s_1, g) \cdot P(o_1|s_1) \cdot P(s_1) \\
& \cdot \prod_{t=3}^{T} P(o_t|s_t) \cdot P(s_t|a_{t-1}, s_{t-1}) \cdot P(a_{t-1}|p_{t-1}, s_{t-1}) \cdot P(p_{t-1}|s_{t-1}, p_{t-2}, g))
\end{aligned}
\tag{2.24}
$$

However, this computation is intractable, as it requires marginalizing over all of these variables. To account for this, the probabilistic SIPS model instead uses a sequential Monte Carlo (SMC) procedure to perform approximate online inference as observations are made. This procedure inverts the generative model described in the previous section to infer the agent's goal at each point in time.

An example of the inferred goal inference posteriors throughout the progression of agent observations in the Doors-Keys-Gems domain can be seen in Figure 2-2. In the Doors-Keys-Gems domain, an agent is in a grid world and aims to obtain one of the gems within the grid. It may need to navigate around walls, pick up keys, and use keys to unlock doors in order to achieve its goal. As shown in the figure, the posterior inferred with SIPS is reasonable to the human eye even when the agent backtracks and the most likely goal changes. The posterior inferred with SIPS is also reasonable even in the case of failed agent planning, such as if an agent uses its only key to open the wrong door and no longer has any means to achieve its goal.

To estimate a posterior using the neural approach, no inversion process needs

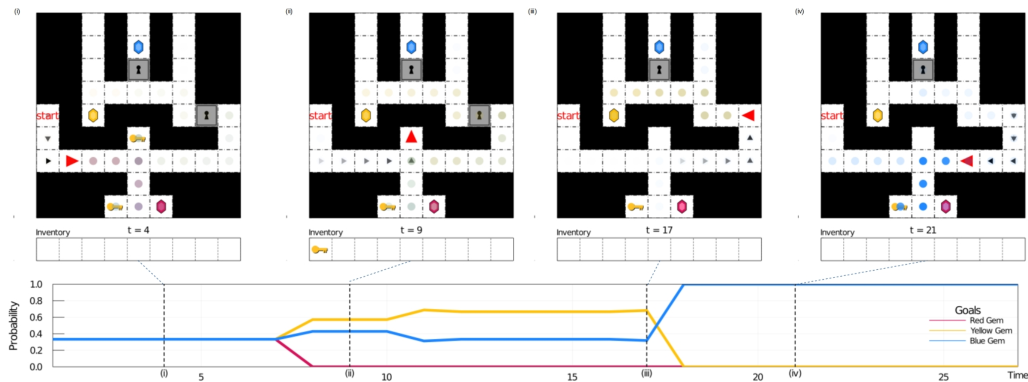

Figure 2-2: A demonstration of the goal inference model predicting which gem the agent aims to obtain. The posterior, as shown by the graph, is approximated online as observations of the agent's actions are made. Replanning allows for limited computation as well as accurate goal inference despite suboptimal planning, as shown in this case. When the agent has reached world state (i), it is unclear which goal the agent is attempting to achieve. When it picks up the key at state (ii), the predicted posterior indicates that one of the gems behind the locks is a more likely goal than the red gem. When the agent proceeds to not pick up the second key and instead unlocks the outer lock at state (iii), the model infers that the yellow gem is the most likely goal. However, by state (iv), when the agent backtracks toward the second key, the model correctly infers that the agent's goal is the blue gem.

to occur. Rather, a neural network is trained in advance on data generated by the same generative model used in the SIPS experiments. Vector representations of the sequence of world states are provided as input to that neural network, and a predicted posterior distribution is returned as output. While more computation might be needed at training time, the neural approach may require less computation than the SIPS approach at run time. Further experiments will be conducted in this thesis to investigate whether this holds true.

Chapter 3

Converting PDDL Programs to Vector Representations

In order to take a neural approach to inferring an agent's goal based on an observation sequence of their actions, the given information must be converted into a form that can be input to a neural network and that provides all the necessary information on the world state. In this research project, the given information is in the form of a PDDL domain that defines a set of predicates and a set of possible actions and their effects, a problem that specifies the given objects and their configurations, and a sequence of actions. Then, the problem is initialized to generate a PDDL state representation that provides information about the types of objects and the predicates that are true of those objects. The given actions are then applied in sequence to transition from the initial state to the subsequent state, and so on. The result is a sequence of PDDL state representations. A significant component of this project consisted of converting such sequences into forms that could be accepted as input to neural networks.

3.1 Technical Challenges

There are multiple domains considered in this project that each involved different predicates and objects. To account for this, multiple approaches were taken to convert the inputs for the different domains.

Blocks World is an example of a domain that consists of objects that can be sorted by their names or identifiers. In the Blocks World domain, an agent is presented with a number of blocks stacked on a table in some configuration, and they must spell some goal word such as "star" or "trash" by stacking the blocks on top of one another. For the purposes of this thesis, the aim was to develop a generalized function to convert from a state in any such domain consisting of sortable objects to a vector representation. A natural representation in this instance is a boolean vector in which each entry represents a ground predicate and is given a value of true if the predicate holds in the state. However, the length of this vector and the order of the entries in the vector become complicated, since they depend on the number of predicates in the domain, the number of arguments and of which type each predicate accepts, and the quantity of each type of object in the problem specification. The ordering of the vector entries must be consistent so that they hold the same meaning across states and problems to allow the neural networks to learn consistent weights.

The Doors-Keys-Gems states, on the other hand, are grid worlds, and thus naturally lend themselves to two-dimensional array representations. The logical representation for representing the grid worlds is a two dimensional array where each entry corresponds to a grid square. Each array entry would be filled with a number indicating the contents of that grid square. However, multiple objects can inhabit the same grid square at once in a Doors-Keys-Gems state, and individually assigning numbers to each possible combination of objects in a grid square was infeasible. This presented a design challenge for developing the code to convert from a Doors-Keys-Gems Prolog-style PDDL state representation to an array suitable for a neural network.

3.2 Conversion of Blocks World

The conversion code from a sequence of Blocks World PDDL states to a format suitable for a neural network was designed to be generalizable to any domain that consists of sortable objects. As such, it needed to be robust to different types of objects, to varying numbers of predicates that accept various numbers of and types

of objects as arguments, and to the presence of any number of fluents.

The chosen neural network input representation was a boolean vector, where each value in the vector indicates the truth or falsehood of a particular ground predicate. The length of the vector representation for a Blocks World state is given by:

$$L = n_{\text{preds}} + n_{\text{fluents}} \tag{3.1}$$

where n_{fluents} is the number of fluents and n_{preds} is the number of ground predicates, as calculated by:

$$n_{\text{preds}} = \sum_{p \in P} \prod_{d \in \textsc{dim}(p)} d \tag{3.2}$$

where P are the generic predicates, and $\textsc{dim}(p)$ is the tuple of dimensions representing the size of the argument space. In particular, the ith element of $\textsc{dim}(p)$ is the number of possible objects that can be substituted in for the ith argument of predicate p, given by:

$$\textsc{dim}(p)_n = \textsc{num-objs-of-type}(t_n) - \textsc{num-objs-of-type-in-sequence}(t_n, a_{0:n-1}) \tag{3.3}$$

where:

$$t_n = \textsc{type-of}(a_n) \tag{3.4}$$

and:

$$a_n = \textsc{args}(p) \tag{3.5}$$

This assumes that the same object cannot be given as two separate arguments to the same predicate, which holds in the case of Blocks World.

To determine which entries correspond to which ground predicates, the predicates were ordered alphabetically. For each predicate, every possible ground predicate was

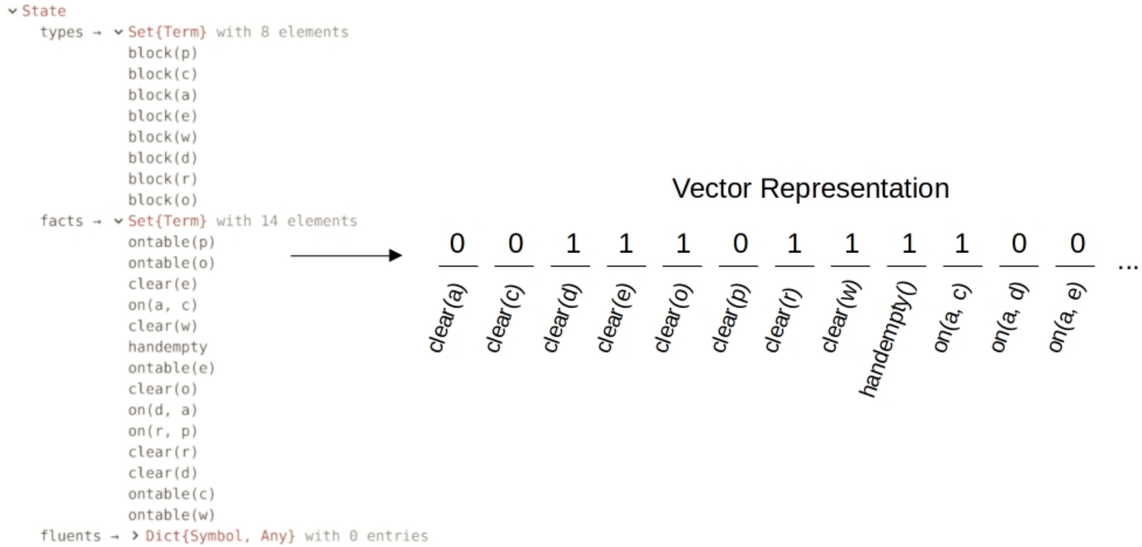

Figure 3-1: Conversion from a Blocks World PDDL state to a vector representation for input to the LSTM. Each ground predicate is represented by a boolean in the vector, where a value of 1 indicates that the ground vector is true. Within the vector, the predicates are sorted alphabetically, and within each predicate's section of the vector, the possible objects are sorted alphabetically for consistency across problems.

generated, accounting for the constraints on the argument object types. Within each ground predicate set corresponding to the same predicate, they were alphabetically ordered primarily by the first argument, secondarily by the second argument, and so on. This lexicographic alphabetical ordering served as the ordering for the boolean vector entries. This conversion process ensures that the entries hold the same meaning across states and problems, and as such the neural networks are able to learn consistent weights. See Figure 3-1 for an example of this conversion from a Blocks World predicate representation to a neural vector representation.

3.3 Conversion of Doors-Keys-Gems

The Doors-Keys-Gems domain is inherently two-dimensional, as it consists of a two-dimensional grid of walls and objects through which the agent must traverse. Therefore, it was desirable to conserve this structure in the state representation that would

be provided to the neural network. Information must also be conveyed about the inventory of the agent, as it can be holding one or more different potential objects including keys or various gems. To allow for both of these types of information to be input to the neural network, a representation consisting of an array and a vector was selected. The conversion to this representation is depicted in Figure 3-2.

The vector portion of the state representation is simply a boolean vector where each entry corresponds to one of the objects in the problem, and the corresponding value is true if the agent is carrying that object and false otherwise. The design of the array representation, however, is more specialized. Each entry in the array corresponds to the grid square in the Doors-Keys-Gems state with the same indices. Initially, a value of one is assigned to each entry in the array. Each type of object is assigned a prime number value. Because all doors, keys, and walls are indistinguishable from one another, each of these object types is assigned a single value. However, since the gems represent the various goals and must be distinguishable by the neural network, each gem is assigned a unique value. Of all the assigned values, the gems are assigned the highest values, so that in every given state the same prime values always indicate the same objects, regardless of how many gems may be in the scene.

Once the array is initialized, the PDDL predicates are iterated over to extract the coordinates of each object. During this iteration, the values in the array are updated to indicate which objects, if any, are located within the corresponding Doors-Keys-Gems grid square. To indicate the presence of an object in a given grid square, the corresponding array value is multiplied by that object's assigned prime value. Thus, due to the uniqueness of primes, any grouping of objects in a single grid square has a unique value.

Different Doors-Keys-Gems problems can also have varied spatial dimensions. If these exact dimensions were used in the array representation, a separate neural network would need to be trained for each dimension size of interest, which would limit generalizability significantly. To avoid this, all array representations are padded up to 16x16. Since agents cannot move through these padded areas as they do not exist in the Doors-Keys-Gems PDDL state, they effectively act as walls. As such, the

Figure 3-2: Conversion from a Doors-Keys-Gems PDDL state to a representation that entails an array and a vector for input to the LSTM. Each type of object is represented by a prime number, and the empty cell is represented by 1. If multiple objects are in the same cell, the primes representing those objects are multiplied together. Due to the uniqueness of prime factorizations, this allows each distinct combination of objects in a cell to be represented by a unique integer. In the array, all keys are represented by the same value, because they are indistinguishable in terms of their usage. However, since the gems represent the distinct goals, each is assigned a distinct prime. The vector elements are booleans representing which items the agent is carrying. The objects within the vector are alphabetically sorted for consistency across problems. The array has a fixed size of 16 by 16 and the vector has a fixed size of 10 so that the same LSTM can be used across many problems with different configurations without needing to train a separate LSTM for each configuration.

padding values used were the same as the values used to indicate walls. Similarly, the vector portions of the state representation must also be padded to account for varying numbers of gems and keys. Each input vector is padded with zeros to a length of ten. This choice of padding value is also logical, since the fact that the padded values are always zero effectively indicates that the agent never holds some object that is not present in the given Doors-Keys-Gems problem.

Chapter 4

Neural Goal Inference via LSTMs

Since the observations of the agent's actions are in the form of a time-ordered sequence of states, a long short-term memory (LSTM) [5] recurrent neural network (RNN) architecture is used. This allows prior states to be remembered by the network for a number of timesteps for more informed inference.

4.1 Mathematical Formulation

The LSTM architecture allows for observations in a sequence to be taken into account for a period of subsequent timesteps, while also preventing misleading observations from excessively influencing the overall neural network output via the use of a forget gate [3]. If the formula for the LSTM forget gate is

$$f_t = \sigma(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}) \tag{4.1}$$

and the formula for the input gate is

$$i_t = \sigma(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}) \tag{4.2}$$

then the cell state is as follows

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + b_{xc} + W_{hc}h_{t-1} + b_{hc}) \tag{4.3}$$

where $b_{xf}, b_{hf}, b_{xi}, b_{hi}, b_{xc}$, and b_{hc} are biases, $W_{xf}, W_{hf}, W_{xi}, W_{hi}, W_{xc}$, and W_{hc} are weight matrices, σ is the sigmoid function, and $\odot$ indicates elementwise matrix multiplication.

Furthermore, the LSTM output equation is as follows.

$$o_t = \sigma(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}) \tag{4.4}$$

where b_{xo} and b_{ho} are biases and W_{xo} and W_{ho} are weight matrices.

Given this, the formula for the LSTM hidden state is then

$$h_t = o_t \odot \tanh(c_t). \tag{4.5}$$

For the activation function on the output, the softmax function is used:

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{4.6}$$

where the function holds for all $x_1 \ldots x_n$, and $n = \text{num_possible_goals}$. Applying this activation function forces the vector of outputs to sum to one, allowing it to be treated as a predicted posterior probability distribution.

4.2 Neural Architecture for Blocks World Domain

The overall architecture for the Blocks World domain is shown in Figure 4-1. It consists of an LSTM layer that feeds into a linear layer. The softmax function is then applied to the output from the linear layer to produce a set of probabilities corresponding to the predicted posterior probability of each possible goal given the sequence of observations received up to that timestep.

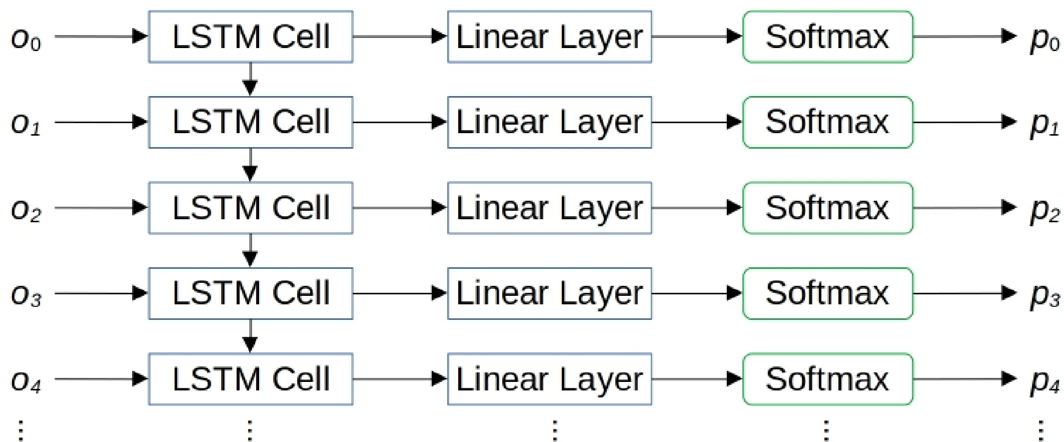

Figure 4-1: Diagram of the architecture of the Blocks World neural network. At each time step, the vector representation of the observed state is input to the LSTM, and the output from the LSTM is input to a linear layer whose output length is equal to the number of potential goals. Finally, the softmax function is applied to the output from the linear layer, and the resulting values are the predicted probabilities of each possible goal being the true goal of the agent given the observations up to that point.

4.3 Neural Architecture for Doors-Keys-Gems Domain

For the Doors-Keys-Gems domain, the relative spatial locations of the objects in the problem are crucially related to the agent's choice of actions to achieve the goal of obtaining a certain gem. For this reason, the first three layers of the Doors-Keys-Gems neural network architecture are convolutional layers. Each of these layers has four output channels to detect various features of the input state. To limit the size of the inputs to the subsequent LSTM layer and to make the network more robust to different problem configurations, both dimensions of the array are downsampled by a factor of two on both the first and third convolutional layers. Then, the outputs are flattened and concatenated with the vector representing the agent's inventory. The newly concatenated vector is input to the LSTM layer, and the output of that layer is given to a linear layer. Finally, the softmax function is applied to the output of this layer to produce a predicted posterior probability distribution on the agent's goal.

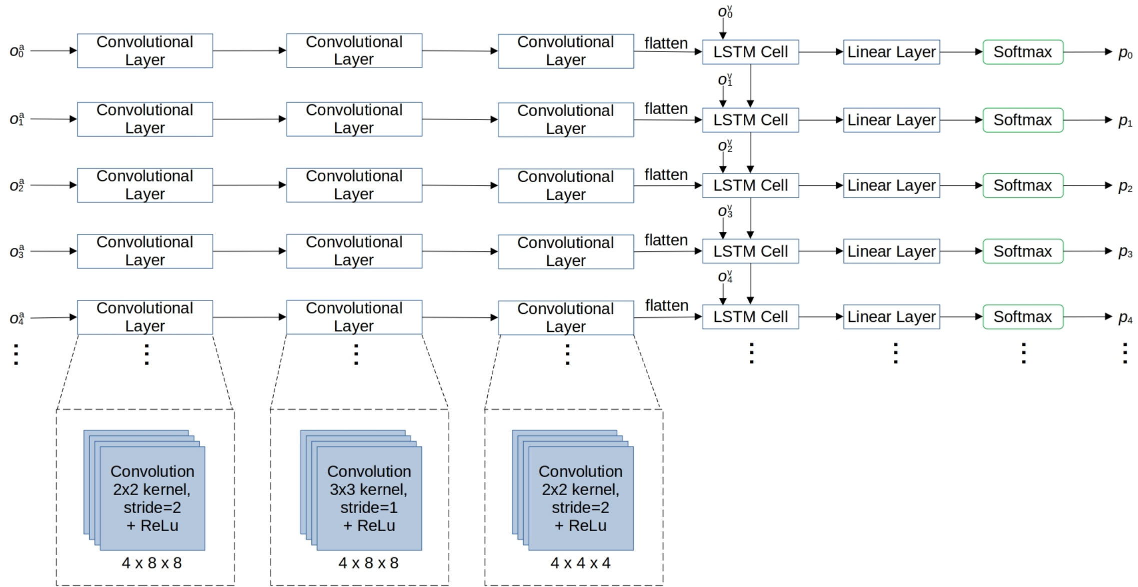

Figure 4-2: Diagram of the architecture of the Doors-Keys-Gems neural network. o_t^a represents the array component of the array and vector state representation at time t, while o_t^v represents the vector component. At each time step, the array portion of the representation of the observed state is input to a series of convolutional layers that each have four channels. These convolutional layers downsample the 16x16 input array to a 4x4 array with 4 channels. This output is then flattened and input to a the LSTM. Like in the Blocks World neural network architecture, this output is input to a linear layer whose output length is equal to the number of possible goals, and the softmax function is applied to the output of the linear layer. The values that result are the predicted probabilities of each possible goal being the true goal of the agent given the observations up to that point.

4.4 Comparing Monte Carlo and Neural Implementations of Bayesian Goal Inference

For each domain, five problems were developed on which to compare the neural approach to the Monte Carlo SIPS approach. Each Blocks World problem had initial states that consisted of the same letters in differing configurations, and each Doors-Keys-Gems problem consisted of an 8x8 or 9x9 grid of independent mazes with three gems and varying numbers of doors and keys. The Blocks World problems each had twenty possible goal words that the agent could aim to spell, and the goals in each Doors-Keys-Gem problem consisted of obtaining any of the three gems. For each of these problem and goal pairs, five training sequences were generated, of which one was an optimal trajectory to achieve the goal and four were suboptimal trajectories. A separate three testing sequences were also generated for each problem and goal pair, one of which was optimal and the other two of which were suboptimal. All of the trajectories were generated using the SIPS generative model with replanning, calling the A^* algorithm as a subroutine.

Once the data generated by the SIPS generative model was used for training the neural networks, the accuracy and computational resource requirements of the neural approach were compared to that of the SIPS approach using the testing dataset. To test the generalizability of the networks for each domain, several sets of experiments were performed. For each domain, the following LSTM experiments were conducted:

- Train and test on one problem - For each problem, a network was trained on only the training data for that problem and tested on only the testing data for that same problem

- Train on only one problem - For each problem, a network was trained on only the training data for that problem and tested on only the testing data for all of the other problems within the same domain

- Leave-one-problem-out - For each problem, a network was trained on all of the training data for all of the other problems in the same domain and tested on

only the testing data for the given problem

- Train and test on all problems - A network was trained on all of the training data for all of the problems within that domain and tested on all of the testing data for all of the problems within the domain

The top-1 accuracy results of these experiments are shown in Table 4.1, where the top-1 accuracy is defined as the proportion of testing examples on which the goal with the highest predicted posterior value at the end of the given quartile is the agent's true goal. In both domains, the top-1 accuracy of the SIPS method in the first quartile of the observations tended to far surpass the results of any of the neural experiments. This may be because the neural approaches place more weight on the proximity of the agent to the various gems in each state representation in the case of Doors-Keys-Gems, or on the number of blocks that are stacked in the proper order for each goal word in the case of Blocks World. However, as time progresses, the accuracy of the neural networks in the experiments where the training and testing data is well-matched tends to improve drastically, and in the third and fourth quartiles, the accuracies of the "train and test on one problem" and the "train and test on all problems" neural experiments are similar to those of SIPS. Compared to the "train on only one problem" and the "leave-one-problem-out" LSTM experiments, though, SIPS consistently yields higher accuracy across all quartiles.

For both domains, within the neural experiment sets, the "train and test on one problem" and "train and test on all problems" LSTM experiments consistently yield the highest accuracies. This is unsurprising, as those two sets of experiments do not require the neural networks to generalize far beyond the data on which they were trained. The "train on only one problem" LSTM experiments consistently produce the lowest accuracies. This is also expected, since these neural networks are only trained on a small dataset from one problem and are then tasked with predicting the agent goals in a variety of other problems with different configurations.

The predicted posterior values of the true goal over all four quartiles for each set of experiments are also shown in Figure 4-3 and Figure 4-4. These figures provide

Table 4.1: The accuracy results of the two approaches on the Blocks World and Doors-Keys-Gems domains. Q1-Q4 represent the different quartiles of each observation trajectory. In the column below each quartile label, the values are the mean top-1 accuracies of each approach, where the top-1 accuracy is the proportion of testing examples on which the goal with the highest predicted posterior value at the end of the quartile is the true goal. The bolded values are the highest mean values in each quartile.

(a) Accuracy results in the Blocks World domain

Blocks World Accuracy

Method	Top-1			
	Q1	Q2	Q3	Q4
SIPS	**0.62**	0.50	0.56	**0.81**
LSTM (train and test on one problem)	0.25	**0.58**	**0.72**	0.79
LSTM (train on only one problem)	0.07	0.14	0.20	0.32
LSTM (leave-one-problem-out)	0.11	0.20	0.27	0.41
LSTM (train and test on all problems)	0.24	0.50	0.59	0.65

(b) Accuracy results in the Doors-Keys-Gems domain

Doors-Keys-Gems Accuracy

Method	Top-1			
	Q1	Q2	Q3	Q4
SIPS	**0.78**	**0.69**	0.78	0.87
LSTM (train and test on one problem)	0.36	0.33	**0.87**	0.84
LSTM (train on only one problem)	0.29	0.31	0.39	0.41
LSTM (leave-one-problem-out)	0.33	0.33	0.33	0.40
LSTM (train and test on all problems)	0.33	0.60	0.80	**0.91**

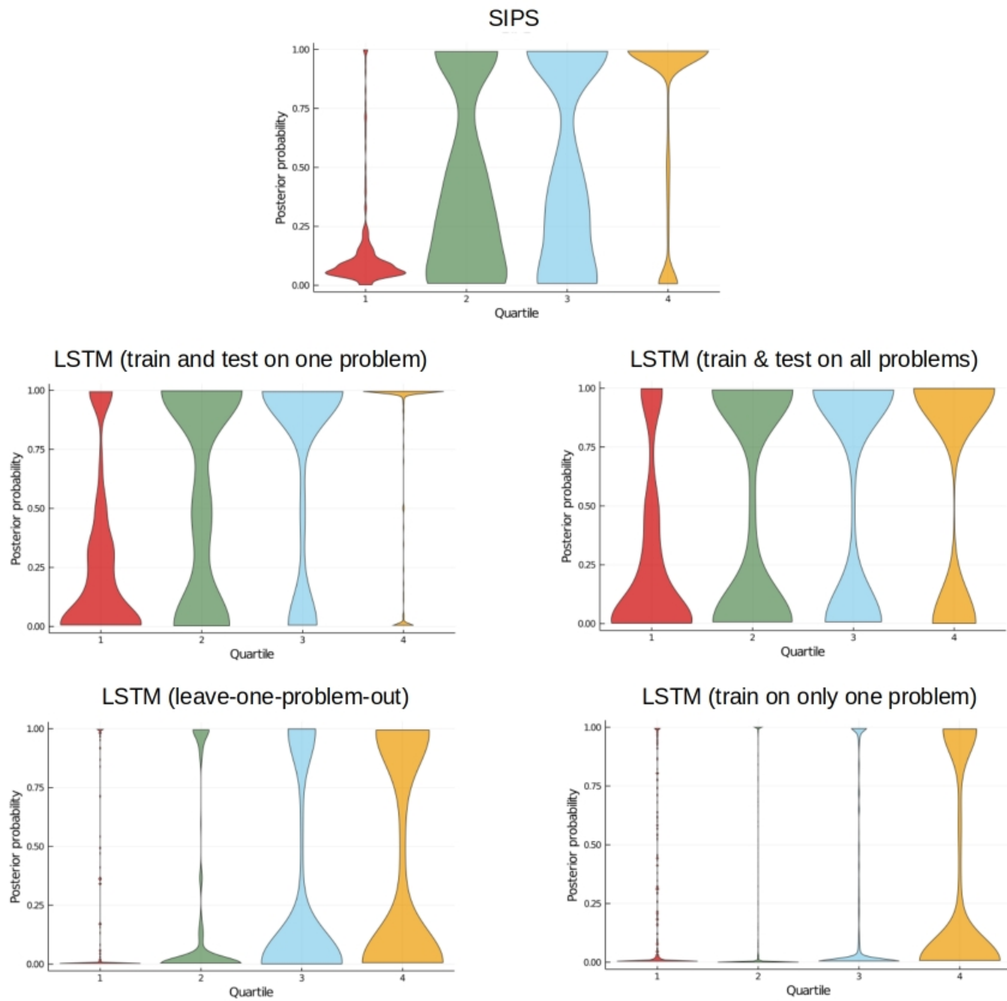

Figure 4-3: Violin plots of the raw true goal predicted posteriors in the Blocks World domain over the four quartiles.

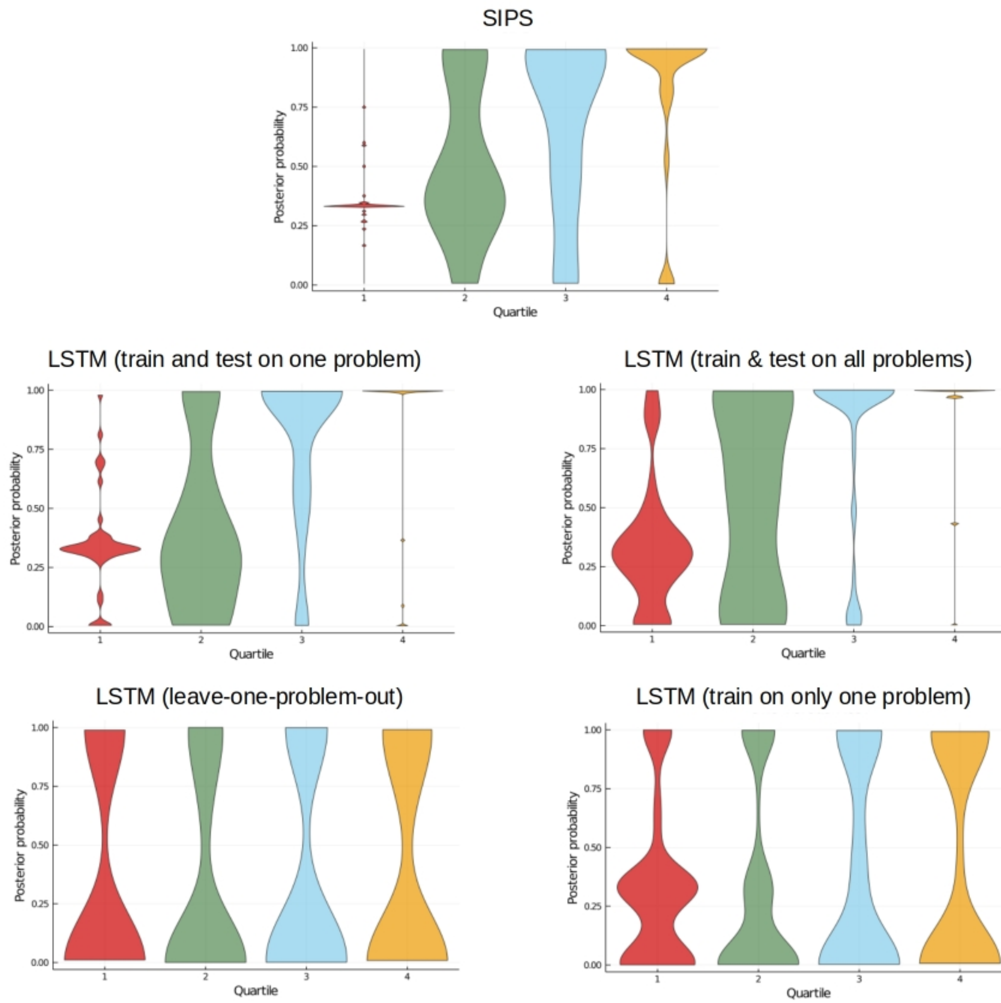

Figure 4-4: Violin plots of the raw true goal predicted posteriors in the Doors-Keys-Gems domain over the four quartiles.

Table 4.2: Computation cost measurements for the experiments in the Blocks World and Doors-Keys-Gems domains. All numbers in the table are the means within the specified experiment sets. The training time metric refers to the length of time it took to train a single neural network within the experiment set. The testing time metric refers to the amount of time required to output predicted posteriors for a single observation sequence in the test set. The amortized time metric conveys the time required to output predicted posteriors for a single observation sequence in the test set, accounting for the offline training time. The training states visited metric refers to the number of states explored during the course of training a single neural network. The amortized states visited refers to the number of states that required visiting to output predicted posteriors for a single observation sequence in the test set, accounting for the states visited in training the network.

(a) Computational resource requirements results in the Blocks World domain

Blocks World Computation Costs

	Training Time (s)	Testing Time (s)	Amortized Time (s)	Training States Visited	Amortized States Visited
SIPS	N/A	190	190	N/A	10000
LSTM (train and test on one problem)	320	$2.4 \cdot 10^{-3}$	5.4	3100000	51000
LSTM (train on only one problem)	200	$1.1 \cdot 10^{-3}$	0.84	3100000	13000
LSTM (leave-one-problem-out)	1700	$2.7 \cdot 10^{-3}$	28	12000000	200000
LSTM (train and test on all problems)	1800	$1.2 \cdot 10^{-3}$	6.0	15000000	51000

(b) Computational resource requirements results in the Doors-Keys-Gems domain

Doors-Keys-Gems Computation Costs

	Training Time (s)	Testing Time (s)	Amortized Time (s)	Training States Visited	Amortized States Visited
SIPS	N/A	22	22	N/A	1800
LSTM (train and test on one problem)	210	$1.1 \cdot 10^{-2}$	23	550000	61000
LSTM (train on only one problem)	173	$2.6 \cdot 10^{-3}$	4.8	550000	15000
LSTM (leave-one-problem-out)	336	$5.2 \cdot 10^{-3}$	37	2200000	240000
LSTM (train and test on all problems)	370	$2.4 \cdot 10^{-3}$	8.2	2700000	61000

some insight into the certainty with which each inference method predicts its goal posteriors. Although it is clear from Table 4.1 that SIPS has relatively high accuracy even in the first quartile, it can be seen in the two violin plots that the true goal posteriors predicted by SIPS are fairly low in the first quartile, indicating a low confidence level. This fits with an intuitive understanding of how humans might predict the goals of one another, as it may be ambiguous early on in an observation sequence which goal the agent is aiming to achieve. The neural approaches, on the other hand, tend to return more extreme results with more confidence, as indicated by the hourglass shapes seen in the neural violin plots. It is of note, though, that often that confidence is misplaced and the predicted posterior values of the true goal are near zero.

The computational resource requirements were measured on these same sets of experiments using a couple different metrics. The computation cost measurements can be found in Table 4.2. Both time metrics and metrics involving the direct measurement of states explored by approach were used. The time metrics show that in almost every case, when amortized over the number of observations tested upon for each neural network, the neural networks are faster than SIPS. The other metric used considers the number of states visited during the processes of testing and training. For SIPS, these states are visited at run time during the plan search process. For the neural approach, these are the states on which the networks are trained, factoring in the number of epochs required to train each network. Unlike the time metric, the amortized number of states visited per test observation sequence solved in all experiments performed was lower for SIPS than for the neural approach. This disparity between the two metrics indicates that SIPS may have the potential to be faster than the neural networks on test sets of this size, if optimizations such as parallelization are added to SIPS. However, even if optimizations are made to SIPS, in cases where one needs to predict agent goals for many observation sequences in the same problem space, due to the amortization of the training costs over all of the testing inputs, the neural approaches will be less resource intensive overall.

Chapter 5

Conclusion

As indicated by the results of the experiments, the accuracy of the neural approach in the last two quartiles of the observations tended to be similar to that of SIPS, in cases where the training data was representative of the testing data. Additionally, the computational resource requirement results showed that without the potential future addition of optimizations to SIPS, the computational resource requirements of SIPS on datasets of this size is consistently higher than that of the neural approach. Finally, the results show that for large testing datasets, amortization makes the neural approach more resource cost effective than SIPS in terms of the number of states visited per observation sequence tested. The combination of these pieces of information implies that currently, in cases where one has a representative training dataset and only requires high accuracy in the later steps of the observation sequences, the neural approach may be equally accurate to and less resource intensive than SIPS.

However, it is often difficult to obtain training datasets that are as similar to the observation state trajectories to which the network would be applied as the training and testing datasets used in the "train and test on one problem" and "train and test on all problems" LSTM experiments where the neural approach performed well. As the "train on only one problem" and "leave-one-problem-out" LSTM experiment results show, SIPS far outperforms the neural approach in cases where the training and testing data are not very well-matched. This indicates that the neural approach does not generalize well, and SIPS provides more accurate results on novel problems

within the PDDL domains. Furthermore, accurate goal inference early on in an agent's trajectory is desirable, and SIPS excels the neural approach in the first, and sometimes second, quartile of the test observation sequences. Thus, perhaps the most promising outcome of the research conducted is the potential to improve the speed of SIPS inference by using a neural proposal based on the networks presented in this thesis.

5.1 Contributions

This research provides a general framework for converting PDDL state representations involving sortable objects to vectors inputtable to neural networks. It also provides a specific framework for converting Doors-Keys-Gems PDDL state representations to vectors and arrays inputtable to neural networks. The research offers an alternative to the SIPS approach for agent goal detection and inverse planning that may be preferable in some circumstances. It also directly compares the two approaches on multiple accuracy and computational resource requirements metrics to provide insight on which approach may be preferable under a given set of circumstances.

5.2 Limitations and Future Work

In terms of optimizing the performance of the neural networks as they are, more intensive testing could be performed to determine the optimal number of epochs used in training each neural network. For the experiments conducted in this work, one number of epochs was selected for each domain and all of the neural networks within that domain were trained with that number of epochs. The selection of the number of epochs was somewhat imprecise in that it was chosen by averaging the estimated number of epochs at which testing accuracy tended to level off in a few randomly selected experimental networks. Training each neural network with an individualized number of epochs may provide improved accuracy for some networks.

Aside from potential limitations in the hyperparameters used in the experiments

conducted in this thesis, one significant limitation this research is that in order to use this Bayesian neural approach on grid-world type domains aside from Doors-Keys-Gems, a more generalized input conversion function would need to be developed that can handle other predicates and object types. Furthermore, the network architectures developed in this project have not been tested for generalization on other domains. As such, the confirmed applications of the developed architecture are somewhat limited.

Another similar limitation is that in the Blocks World domain, a single neural network cannot be trained and tested on observation sequences with varied numbers of objects, since the input dimensions would differ. One potential solution to allow this would be to use graph neural networks, which accept variable-sized graphs as input [1]. This approach would allow for the conversion of a relational state representation with a variable number of objects, like a PDDL representation, into a labeled graph where each vertex would be an object, each predicate that involves one variable would be a vertex label, and each predicate involving two variables would be a labeled edge between the appropriate vertices. This graph could then be provided to a graph neural network [4, 12] to learn a vector representation for each vertex and edge. These vectors could be processed via some method such as pooling or averaging to obtain a single output vector that could be passed to the neural architecture described in this thesis.

The approaches taken in this thesis also prevent a single network from being trained on problems with varying numbers of goals. One strategy to mitigate this limitation might be to use the aforementioned graph neural network approach to convert each PDDL goal specification to a vector representation. Then, a function can be learned that takes as input a hidden state at time t obtained from the RNN and the goal vector representation and outputs a new vector. Each of these vectors can be provided to one of many simple architectures that map vectors to scalars to obtain a scalar score for each goal at each timestep. Finally, the softmax function can be applied to these scalar values to obtain a probability distribution over all of the goals. This approach allows for a direct relationship between a sequence of observation representations and any possible goal to be learned, rather than a relationship between a sequence of observation representations and a fixed number of arbitrarily assigned

goal indices.

Another limitation is that the difference in accuracy between the SIPS approach and the neural approach on optimal versus suboptimal sequences where agents potentially fail to achieve their goals is unclear, as that was not specifically investigated. A detailed comparison on this would need to be conducted to determine which Bayesian inference approach would provide better results under each of these circumstances.

There remain many interesting questions regarding the tradeoffs between the generality of the model-based inference approach and the performance and speed of the neural inference approach. Pursuing future work in the aforementioned directions may help to answer some of these questions. Some of the improvements suggested in this thesis may aid in finding an approach to goal inference with satisfactory accuracy and computational resource requirements, whether through improving one of the two individual inference methods analyzed through this research, or by using the two methods in conjunction to increase the speed of SIPS inference by using a neural proposal.

Bibliography

[1] Xavier Bresson and Thomas Laurent. An experimental study of neural networks for variable graphs. *ICLR 2018 Workshop*, 2018.

[2] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: A general-purpose probabilistic programmingsystem with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '19).*

[3] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[4] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735.

[6] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl - the planning domain definition language, 1998.

[7] Bernard Michini and Jonathan P How. Improving the efficiency of bayesian inverse reinforcement learning. In *2012 IEEE International Conference on Robotics and Automation*, page 3651.

[8] Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, page 663.

[9] Neil C Rabinowitz, Frank Perbet, H Francis Song, Chiyuan Zhang, SM Eslami, and Matthew Botvinick. Machine theory of mind. *arXiv preprint arXiv:1802.07740*, 2018.

[10] Miquel Ramírez and Hector Geffner. Plan recognition as planning. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[11] Daniel Ritchie, Paul Horsfall, and Noah D Goodman. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016.

[12] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29:2244–2252, 2016.

[13] Tan Zhi-Xuan, Jordyn L. Mann, Tom Silver, Joshua B. Tenenbaum, and Vikash K. Mansinghka. Online bayesian goal inference for boundedly-rational planning agents. To be published in NeurIPS proceedings, 2020.

[14] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, page 1433.