Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Symbolic Reinforcement Learning using Inductive Logic Programming

Author:
Kiyohito Kunii

Supervisor:
Prof. Alessandra Russo
Mark Law
Ruben L Vereecken

Submitted in partial fulfillment of the requirements for the MSc degree in MSc in Computing Science of Imperial College London

June 2018

Abstract

Your abstract.

Acknowledgments

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There have been successful applications of deep reinforcement learning (DRL) in a number of domains, such as video games [**?**], the game of Go [**?**] and robotics [**?**]. However, there are still a number of issues to overcome with this method. First, it requires large dataset for training the model, and the learning is very slow and requires significant amount of computation. Second, it is considered to be a black-box, meaning that the decision making process is unknown to the human user and therefore lacks explanation of the decision making. Third, there is no thought process to the decision making, such as understanding relational representations or planning. To tackle these problems, researchers have explored several different approaches. One of the methods is to incorporate symbolic representations into the system [**?**]. This approach is promising and shows a potential.
In this paper, we extend this symbolic representation approach and explore the potential of symbolic machine learning to solve the above issues. There are several advantages of symbolic machine learning. First of all, the decision making mechanism is understandable by humans rather than being black-box. Second, it resembles how humans reason. Similar to reinforcement learning, there are some aspects of trial-and-error in human learning, but humans exploit reasonings to efficiently learn about their surrounding or situations. They also effectively use previous experience (e.g background knowledge) when encountering similar situations. Finally, the recent advance of Inductive Logic Programming (ILP) research has enabled us to apply ILP in more complex situations and there are a number of new algorithms based on Answer Set Programmings (ASPs) that work well in non-monotonic scenarios.
Particularly since [**?**], there have been several researches that further explored the incorporation of symbolic reasoning into RL, but the combining of ILP and RL has not been explored. Because of the recent advancement of ILP and RL, it is natual to consider that a combination of both approaches would be the next field to explore.
In this paper, our objective is to explore the incorporation of ILP into RL using Inductive Learning of Answer Set Programs (ILASP), which is a state-of-art ILP method that can be applied to incomplete and more complex environments.
TODO Update this
This background report will be part of the final report and is organised as follows: In Chapter **??**, the background of inductive logic programming and reinforcement learning necessary for this paper are described. Chapter **??** discusses previous re-

search on relevant approach. Chapter **??** shows the tentative architecture of our new approach, using ILASP to generate a model of the environment. We also discuss some of the issues we currently face with the architecture and plan the implementation. Finally the ethics checklist is provided in Chapter **??**.

Chapter 2

Background

This chapter introduces necessary background of Inductive Logic Programming (Section **??**) and Reinforcement Learning (Section **??**), which provide the foundations of our research.

2.1 Inductive Logic Programming (ILP)

Inductive Logic Programming (ILP) is a subfield of machine learning research area aimed at the intersection between machine learning and logic programming [**?**]. The purpose of ILP is to inductively derive a hypothesis H that is a solution of a learning task, which coveres all positive examples and none of negative examples, given a hypothesis language for search space and cover relation [**?**]. ILP is based on learning from entailment, as shown in Equation **??**.

$$B \wedge H \models E \tag{2.1}$$

where E contains all of the positive examples (E^+) and none of the negative examples (E^-). One of the advantage of ILP over statistical machine learning is that the hypothesis that an agent learnt can be easily understood by a human, as it is expressed in first-order logic, making the learning process more transparent rather than black-box. One of the limitations of ILP is learning efficiency and scalability. There are usually thousands or more examples in many real-world examples. Scaling ILP task to cope with large examples is a challenging task [**?**].
In this section, we briefly introduce foundation of Answer Set Programming (ASP) and inductive learning frameworks.

2.1.1 Stable Model Semantics

Having defined the syntax of clausal logic, we now introduce its semantics under the context of Stable Model. The semantics of the logic is based on the notion of interpretation, which is defined under a *domain*. A domain contains all the objects that exist. In logic, it is convention to use a special interpretations called *Herbrand interpretations* rather than general interpretations.

Definition 2.1. *Herbrand Domain* (a.k.a *Herbrand Universe*) of clause sets *Th* is the set of all ground terms that are constants and function symbols appeared in *Th*.

Definition 2.2. *Herbrand Base* of *Th* is the set of all ground predicates that are formed by predicate symbols in *Th* and terms in the Herbrand Domain.

Definition 2.3. *Herbrand Interpretation* of a set of definite clauses *Th* is a subset of the Herbrand base of *Th*, which is a set of ground atoms that are true in terms of interpretation.

Definition 2.4. *Herbrand Model* is a Herbrand interpretation if and only if a set *Th* of clauses is satisfiable. In other words, the set of clauses *Th* is unsatisfiable if no Herbrand model was found.

Definition 2.5. *Least Herbrand Model* (denoted as *M(P)*) is an unique minimal Herbrand model for definite logic programs. The Herbrand Model is a minimum Herbrand model if and only if none of its subsets is an Herbrand model.

For normal logic programs, there may not be any least Herbrand Model.

example 2.1.1. (Herbrand Interpretation, Herbrand Model and M(P))

$$\text{P} = \begin{cases} p(X) \leftarrow q(X) \\ q(a). \end{cases} \quad \text{HD} = \{ \text{ a } \}, \text{HB} = \{ \text{ q(a), p(a) } \}$$

where HD is Herbrand Domain and HB is Herbrand Base. Given above, there are four Herbrand Interpretations = $\langle$ {q(a)}, {p(a)}, {q(a), p(a)}, {} $\rangle$, and one Herbrand Model (as well as M(P)) = {q(a), p(a)}

Definite Logic Program is a set of definite rules, and a *definite rule* is of the form $h \leftarrow a_1, ..., a_n$. h and $a_1, ..., a_n$ are all atoms. h is the *head* of the rule and $a_1, ..., a_n$ are the *body* of the rule. *Normal Logic Program* is a set of normal rules, and a normal rule is of the form $h \leftarrow a_1, ..., a_n$, *not* $b_1, ...,$ *not* b_n where h is the head of the rule, and $a_1, ..., a_n, b_1, ..., b_n$ are the body of the rule (both the head and body are all atoms).
To solve a normal logic program *Th*, the program P needs to be grounded. The *grounding* of *Th* is the set of all clauses that are c $\in$ *Th* and variables are replaced by terms in the Herbrand Domain.

Definition 2.6. The algorithm of grounding starts with an empty program Q = { } and the relevant grounding is constructed by adding to each rule R to Q such that

- R is a ground instance of a rule in P.
- Their positive body literals already occurs in the in the of rules in Q.

The algorithm terminates when no more rules can be added to Q.

example 2.1.2. Grounding

$$\text{P} = \begin{cases} q(X) \leftarrow p(X). \\ p(a). \end{cases}$$

ground(P) in this example is {p(a), q(a)}.

Not only the entire program needs to be grounded in order for an ASP solver to work, but also each rule must be *safe*. A rule *R* is safe if every variable that occurs in the head of the rule occurs at least once in body^{+}(R) . Since there is no unique least Herbrand Model for a normal logic program, Stable Model of a normal logic program was defined in [**?**]. In order to obtain the Stable Model of a program P, P needs to be converted using *Reduct* with respect to an interpretation X.

Definition 2.7. The *reduct* of P with respect to X can be constructed such that

- If the body of any rule in P contains an atom which is not in X, those rules need to be removed.
- All default negation atoms in the remaining rules in P need to be removed.

example 2.1.3. Reduct

P = $\begin{cases} p(X) \leftarrow not\ q(X). \\ q(X) \leftarrow not\ p(X). \end{cases}$, X = {p(a), q(b)}
Where X is a set of atoms. ground(P) is
p(a) ← not q(a).
p(b) ← not q(b).
q(a) ← not p(a).
q(b) ← not p(b).

The first step removes p(b) ← not q(b). and q(a) ← not p(a).
p(a) ← not q(a).
q(b) ← not p(b).

The second step removes negation atoms from the body.
Thus reduct P^{x} is $(ground(P))^{x}$ = {p(a), q(b).}

A Stable Model of P is an interpretaiton X if and only if X is the unique least Herbrand Model of $ground(P)^{x}$ in the logic program.

2.1.2 Anwer Set Programming (ASP) Syntax

Definition 2.8. Answer set of normal logic program P is a Stable Model, and Answer Set Programming (ASP) is a normal logic program with extensions: constraints, choice rules and optimisation statements. ASP program consists of a set of rules, where each rule consists of an atom and literals.

A *constraint* of the program P is of the form ← a_1, ..., a_n, *not* b_1, ..., *not* b_n, where the rule has an empty head. The constraint filters any irrelevant answer sets. When computing $ground(P)_x$, the empty head becomes ⊥, which cannot be in the answer sets. There are two types of constraints: *hard constraints* and *soft constraints*. Hard constraints are strictly satisfied, whereas soft constraints may not be satisfied but the sum of the violations should be minimised when solving ASP.

A *choice rule* can express possible outcomes given an action choice, which is of the form $l\{h_1,...,h_m\}u \leftarrow a_1, ..., a_n$, not $b_1, ...,$ not b_n where l and u are integers and h_i for $1 \leq i \leq m$ are atoms. The head is called *aggregates*.
Optimisation statement is useful to sort the answer sets in terms of preference, which is of the form *#minimize[a_1=w_1,...a_n=w_n]* or *#maximize[a_1=w_1,...a_n=w_n]* where w_1,..., w_n is integer weights and a_1,...,a_n is ground atoms. ASP solvers compute the scores of the weighted sum of the sets of ground atoms based on the true answer sets, and find optimal answer sets which either maximise or minimise the score.
Clingo is one of the modern ASP solvers that executes the ASP program and returns answer sets of the program ([**?**]), and we will use *Clingo* for the implementation of this research.

2.1.3 ILP under Answer Set Semantics

There are several ILP non-monotonic learning frameworks under the answer set semantics . We first introduce two of them: *Cautious Induction* and *Brave Induction* ([**?**]), which are foundations of *Learning from Answer Sets* discussed in Section **??**, a state-of-art ILP framework that we will use for our research. (for other non-monotonic ILP frameworks, see [**?**], [**?**], [**?**] and [**?**]).

Cautious Induction

Cautious Induction task [1] is of the form $\langle$ B, E^+, E^- $\rangle$, where B is the background knowledge, E^+ is a set of positive examples and E^- is a set of negative examples.
$H \in ILP_{cautious} \langle$ B, E^+, E^- $\rangle$ if and only if there is at least one answer set A of B $\cup$ H (B $\cup$ H is satisfiable) such that for every answer set A of B $\cup$ H:

1. $\forall$ e $\in E^+$: e $\in$ A
2. $\forall$ e $\in E^-$: e $\notin$ A

example 2.1.4. Cautious Induction

$$B = \begin{cases} exercises \leftarrow not\ eat\_out. \\ eat\_out \leftarrow exercises. \end{cases} \qquad E^+ = \{\text{tennis}\},\ E^- = \{\text{eat\_out }\}$$

One possible $H \in ILP_{cautious}$ is {tennis$\leftarrow$ exercises, $\leftarrow$ not tennis }.

The limitation of Cautious Induction is that positive examples must be true for all answer sets and negative examples must not be included in any of the answer sets. These conditions may be too strict in some cases, and Cautious Induction is not able to accept a case where positive examples are true in some of the answer sets but not all answer sets of the program.

[1] This is more general definition of Cautious Induction than the one defined in [**?**], as the concept of negative examples was not included in the original definition.

example 2.1.5. Limitation of Cautious Induction

$$\text{B} = \begin{cases} 1\{situation(P, awake), situation(P, sleep)\}1 \leftarrow person(P). \\ person(john). \end{cases}$$

Neither of *situation(john, awake)* nor *situation(john, sleep)* is false in all answer sets. In this example, it only returns person(john). Thus no examples could be given to learn the choice rule.

Brave Induction

Brave Induction task is of the form $\langle$ B, E^+, E^- $\rangle$ where, B is the background knowledge, E^+ is a set of positive examples and E^- is a set of negative examples. H $\in$ ILP_{brave} $\langle$ B, E^+, E^- $\rangle$ if and only if there is at least one answer set A of B $\cup$ H such that:

1. $\forall$ e $\in$ E^+ : e $\in$ A

2. $\forall$ e $\in$ E^- : e $\notin$ A

example 2.1.6. Brave Induction

$$\text{B} = \begin{cases} exercises \leftarrow not\ eat\_out. \\ tennis \leftarrow holiday \end{cases} \qquad E^+ = \{\text{tennis}\},\ E^- = \{\text{eat\_out }\}$$

One possible H $\in$ ILP_{brave} is {tennis}, which returns {tennis, holidy, exercises} as answer sets.

The limitation of Brave Induction that it cannot learn constraints, since the above conditions for the examples only apply to at least one answer set A, whereas constrains rules out all answer sets that meet the conditions of the Brave Induction.

example 2.1.7. Limitation of Brave Induction (Example)

$$\text{B} = \begin{cases} 1\{situation(P, awake), situation(P, sleep)\}1 \leftarrow person(P). \\ person(C) \leftarrow super\_person(C). \\ super\_person(john). \end{cases}$$

In order to learn the constraint hypothesis H = { $\leftarrow$ not situation(P, awake), super_person(P)}, it is not possible to find an optimal solution.

2.1.4 Inductive Learning of Answer Set Programs (ILASP)

Learning from Answer Sets (LAS)

Learning from Answer Sets (LAS) was developed in [**?**] to faciliate more complex learning tasks that neither Cautious Induction nor Brave Induction could learn. Examples used in LAS are *Partial Interpretations*, which are of the form $\langle e^{inc}, e^{exc} \rangle$. (called *inclusions* and *exclusions* of *e* respectively). A Herbrand Interpretation extends a partial interpretation if it includes all of e^{inc} and none of e^{exc}. LAS is of the form $\langle B, S_M, E^+, E^- \rangle$, where B is background knowledge, S_M is hypothesis space, and E^+ and E^- are examples of positive and negative partial interpretations. S_M consists of a set of normal rules, choice rules and constraints. S_M is specified by *language bias* of the learning task using *mode declaration*. Mode declaration specifies what can occur in a hypothesis by specifying the predicates, and consists of two parts: *modeh* and *modeb*. *modeh* and *modeb* are the predicates that can occur in the head of the rule and body of the rule respectively. Language bias is the specification of the language in the hypothesis in order to reduce the search space for the hypothesis.

Definition 2.9. Learning from Answer Sets (LAS)
Given a learning task T, the set of all possible inductive solutions of T is denoted as $ILP_{LAS}(T)$, and a hypothesis H is an inductive solution of $ILP_{LAS}(T)$ $\langle B, S_M, E^+, E^- \rangle$ such that:

1. $H \subseteq S_M$
2. $\forall e \in E^+ : \exists A \in$ Answer Sets$(B \cup H)$ such that A extends e
3. $\forall e \in E^- : \nexists A \in$ Answer Sets$(B \cup H)$ such that A extends e

Inductive Learning of Answer Set Programs (ILASP)

Inductive Learning of Answer Set Programs (ILASP) is an algorithm that is capable of solving LAS tasks, and is based on two fundamental concepts: *positive solutions* and *violating solutions*.
A hypothesis H is a positive solution if and only if

1. $H \subseteq S_M$
2. $\forall e^+ \in \exists A \in$ Answer Sets$(B \cup H)$ such that A extends e^+

A hypothesis H is a violating solution if and only if

1. $H \subseteq S_M$
2. $\forall e^+ \in E^+ \exists A \in$ Answer Sets$(B \cup H)$ such that A extends e^+
3. $\exists e^- \in E^- \exists A \in$ Answer Sets$(B \cup H)$ such that A extends e^-

Given both definitions of positive and violating solutions, ILP_{LAS} $\langle B, S_M, E^+, E^- \rangle$is positive solutions that are not violating solutions.

A Context-dependent Learning from Answer Sets

Context-dependent learning from ordered answer sets ($ILP^{context}_{LOAS}$) is a further generalisation of ILP_{LOAS} with *context-dependent examples* [**?**] Context-dependent examples are examples that each unique background knowledge (context) only applies to specific examples. This way the background knowledge is more structured rather than one fixed background knowledge that are applied to all examples. Formally, partial interpretation is of the form $\langle$ e, C $\rangle$ (called *context-dependent partial interpretation (CDPI)*), where *e* is a partial interpretation and C is called *context*, or an ASP program without weak constraints. A *context-dependent ordering example (CDOE)* is of the form $\langle \langle e_1, C_1 \rangle, \langle e_2, C_2 \rangle \rangle$, which is a pair of CDPI. An APS program P *bravely respects o* if and only if

1. $\exists \langle A_1, A_2 \rangle$ such that $A_1 \in$ Answer Sets$(P \cup C_1)$, $A_2, \in$ Answer Sets$(P \cup C_2)$, A_1 extends e_1, A_2 extends e_2 and $A_1 \prec_P A_2$

Similarly, an APS program P *cautiously* respects *o* if and only if

1. $\forall \langle A_1, A_2 \rangle$ such that $A_1 \in$ Answer Sets$(P \cup C_1)$, $A_2, \in$ Answer Sets$(P \cup C_2)$, A_1 extends e_1, A_2 extends e_2 and $A_1 \prec_P A_2$

$ILP^{context}_{LOAS}$ task is of the form $T = \langle B, S_M, E_+, E_-, O^b, O^c \rangle$ where O^b and O^c are brave and cautious orderings respectively, which are sets of ordering examples over set of positive partial interpretations E^+. A hypothesis H is an inductive solution of T if and only if

1. $H \subseteq S_M$ in $ILP^{context}_{LOAS}$

2. $\forall \langle e, C \rangle \in E^+, \exists A \; \exists A \in$ Answer Sets $(B \cup C \cup H)$ such that A extends e

3. $\forall \langle e, C \rangle \in E^-, \nexists A \; \exists A \in$ Answer Sets $(B \cup C \cup H)$ such that A extends e

The two main advantages of adding contex-dependent are that it increases the efficiency of learning tasks, and more expressive structure of the background knowlege to particular examples. These features will be useful when a game agent is in two different environmets as discussed in Section **??**.

2.2 Reinforcement Learning (RL)

Reinforcement learning (RL) is a subfield of machine learning regarding how an agent behaves in an environment in order to maximise its total reward. As shown in Figure **??**, the agent interacts with an environment, and at each time step the agent takes an action and receives observation, which affects the environment state and the reward (or penalty) it receives as the action outcome. In this section, we briefly introduce the background in RL necessary for our research.

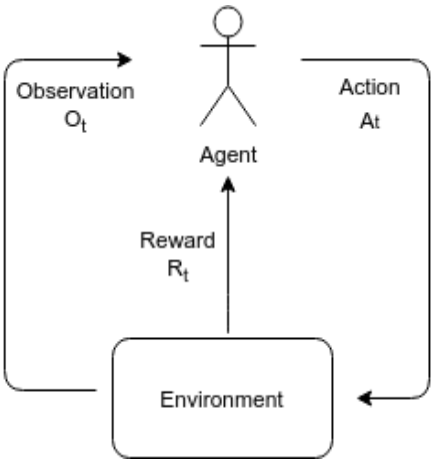

Figure 2.1: Agent and Environment

2.2.1 Markov Decision Process (MDP)

An agent interacts with an environment at a sequence of discrete time step, which is part of the sequential history of observations, actions and rewards. The sequential history is formalised as $H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$. A *state* is a function of the history $S_t = f(H_t)$, which determines the next environment. A state S_t is said to have *Markov property* if and only if $P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, ..., S_t]$. In other words, the probability of reaching S_{t+1} depends only on S_t, which captures all the relevant information from the earilier history ([**?**]). When an agent must make a sequence of decision, the sequential decision problem can be formalised using *Markov decision process (MDP)*. MDP formaly represents a fully observable environment of an agent for RL.
A MDP is of the form $\langle S, A, T_a, R_a, \gamma \rangle$ where:

- S is the set of finite states that is observable in the environment.
- A is the set of finite actions taken by the agent.
- $T_a(s, s')$ is a state transition in the form of probability matrix $Pr(S_{t+1} = s' \mid s_t = s, a_t = a)$, which is the probablity that action *a* in state *s* at time *t* will result in state s' at time *t+1*.
- R is a reward function $R_a(s, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$, the expected immediate reward that action *a* in state *s* at time *t* will return.
- γ is a discount factor $\gamma \in [0,1]$, which represents the preference of the agent for the present reward over future rewards.

2.2.2 Policies and Value Functions

Value functions estimate the expected return, or expected future rewarded, for a given action in a given state. The expected reward for an agent is dependent on

the agent's action. The state value function $v_\pi(s)$ of an MDP under a policy π is the expected return starting from state *s*, which is of the form:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{2.2}$$

where $G_t = R_{t+1} + \gamma R_{t+2} + ... \gamma^{T-1} R_T$, or the total discounted reward from *t*.
The optimal state-value function $v^*(s)$ maximises the value function over all policies in the MDP, which is of the form:

$$v^*(s) = \max_\pi v_\pi(s) \tag{2.3}$$

The optimal action-value function $q^*(s)$ maximises the action-value function over all policies in the MDP, which is of the form:

$$q^*(s,a) = \max_\pi q_\pi(s,a) \tag{2.4}$$

A solution to the sequential decision problem is called a *policy* π, a sequence of actions that leads to a solution. An optimal policy achieves the optimal value function (or action-value function), and it can be computed by maximising over the optimal value function (or action-value function).
TODO BELLMAN OPTIMALITY EQUATION
TODO Value iterations

2.2.3 Model-based and Model-free Reinforcement Learning

TODO delte dyna and focus more on model-based approach
A model M is a representation of an environment that an agent can used to understand how the environment should look like . Model-based learning is that the agent learns the model and plan a solution using the learnt model. Once the agent learns the model, the problem to be solved becomes a planning problem for a series of actions to achieve the agent's goal. Most of the reinforcement learning problems are model-free learning, where M is unknown and the agent learns to achieve the goal by solely interacting with the environment. Thus the agent knows only possible states and actions, and the transition state and reward probability functions are unknown.
The performance of model-based RL is limited to optimal policy given the model M. In other words, when the model is not a representation of the true MDP, the planning algorithms will not lead to the optimal policy, but a suboptimal policy.
One algorithm which combine both aspects of model-based and model-free learning to solve the issue of sub-optimality is called Dyna ([**?**]), which is shown in Figure **??**.
Dyna learns a model from real experience and use the model to generate simulated experience to update the evaluation functions. This approach is more effective because the simulated experience is relatively easy to generate compared building up real experience, thus less iterations are required.

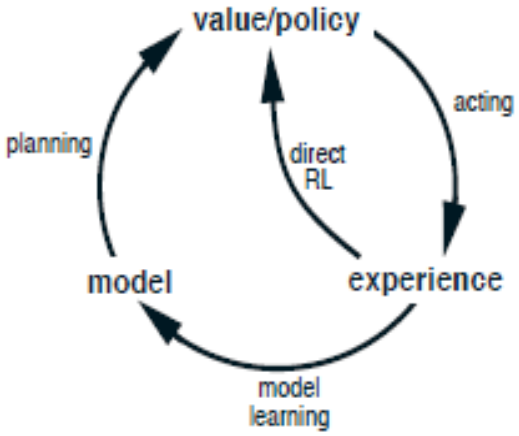

Figure 2.2: Relationships among learning, planning and acting

2.2.4 Temporal-Difference (TD) Learning

To solve a MDP, one of the approaches is called *Temporal-Difference (TD) Learning*. TD is an online model-free learning and learns directly from episodes of imcomplete experiences without a model of the environment. TD updates the estimate by using the estimates of value function by bootstrap, which is formalised as

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.5}$$

where $R_{t+1} + \gamma\ V(S_{t+1})$ is the target for TD update, which is biased estimated of $v_\pi\ (S_t)$, and $\delta = R_{t+1} + \gamma\ V(S_{t+1})$ - $V(S_t)$ is called TD error, which is the error in $V(S_t)$ available at time t+1. Since TD methods only needs to know the estimate of one step ahead and does not need the final outcome of the episodes, it can learn online after every time step. TD also works without the terminal state, which is the goal for an agent. TD(0) is proved to converge to v_π in the table-based case (non-function approximation). However, because bootstraping updates an estimate for an estimate, some bias are inevitable.
Q-learning is off-policy TD learning defined in [**?**], where the agent only knows about the possible states and actionns. The transition states and reward probability functions are unknown to the agent. It is of the form:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma max(a+t)Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.6}$$

where α is the learning rate, γ is a discount rate between 0 and 1. The equation is used to update the state-action value function called Q function. The function Q(S,A) predicts the best action A in state S to maximise the total cumulative rewards.

$$Q(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|s_t, a_t] \tag{2.7}$$

Q-learning is guaranteed to converge to a optimal policy in a fininte tabulara representation. Paper Jaakkola et al. 1993
The optimal Q-function Q^*(s,a) is directly approximated by the learned action-value function Q.

Algorithm 2 XXXX

1: **procedure** ILASP(RL) (B AND E)
2: Initialise Q(s,a) arbitrarily
3: Repeat (for each episode)
4: Choose a from s using policy derived from Q (e.g, epsilon-greedy)
5: Take action a, observe r, s′

Q-learning learns the value of its deterministic greedy policy from the experience and gradually converge to the optimal Q-function. It also explored following ϵ-*greedy policy*, which is a stochastic greedy policy, but with the probability of ϵ, the agent chooses an action randomly instea of the greedy action.

2.2.5 Function Approximation

Q-learning with tarbular method works when every state has Q(s,a). In case of very large MDPs, however, it may not be possible to represent all states with a lookup table. For example, robot arms has a continuous states in 3D dimentional space.
These problems motivate the use of function approximation, which estimates value function with function approximation. Not only it is represented in tabular form, but also in the form of a parameterized function with weight vector w $\in \mathbb{R}^{\mathrm{d}}$ where $\mathbb{R}^{\mathrm{d}}$ is XXX
Unlike Q-table, changing one weight updates the estimated value of not only one state, but many states, and this generalisation makes it more flexible to apply different scenarios that tabular approach could not be applied.
The reason we are introduing this function approximation is not because we will use it in our new algorithm, but for the benchmark that we compare our algorithm with.

The Prediction Objective ($\overline{VE}$)

With function approximation, an update at one state changes many other states, and therefore the values of all states will not be exactly accurate, and there is a tradeoff among states as to which state we make it more accurate, while other might be less accurate.
The error in a state s is the squeare of the difference between the approximate value $\hat{v}$(s,w) and the true value v_{π}(s). The objective function can be defined by weighting it over the statespace by μ, the *Mean Squared Value Error*, denoted $\overline{VE}$.

$$\overline{VE}(w) \doteq \sum_{s \in S} \mu(s) \big[v_{\pi}(s) - \hat{v}(s,w) \big]^2. \tag{2.8}$$

Stochastic gradient descent (SGD)

Stochastic gradient descent methods are commonly used to learn function approximation in value prediction, which works well for online reinforcement learning.

TODO EXPLAIN ONLINE VS OFFLINE LEARNING

$$w \doteq \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \tag{2.9}$$

and $\hat{v}$(s,w) is a differentiable function of w for all s $\in$ S.
minimize the $\overline{VE}$ on the observed examples. *Stochastic gradient-descent (SGD)* adjusts the weights vector by a fraction of alpha in the direction what will reduce the error on that example the most. Formally, it is defined as

$$\begin{aligned} w_{t+1} &\doteq w_t - \frac{1}{2}\alpha \bigtriangledown \big[v_\pi(S_t) - \hat{v}(S_t, w_t)\big]^2. \\ &= w_t - \alpha\big[v_\pi(S_t) - \hat{v}(S_t, w_t)\big] \bigtriangledown \hat{v}(S_t, w_t). \end{aligned} \tag{2.10}$$

where α is step-size,
The dradient of J(w) is defined as

$$\bigtriangledown_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix} \tag{2.11}$$

$$w_{t+1} \doteq w_t + \alpha\big[U_t - \hat{v}(S_t, w_t)\big]x(S_t) \tag{2.12}$$

Linear Value Function Approximation

Formally,

$$\hat{q}(s, a) \approx q_\pi(s, a) \tag{2.13}$$

Represent state by a *feature vector*

$$x(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix} \tag{2.14}$$

Use SGD updates with linear function approximation. The gradient of the approximate value function with respect to w is
Add proof here

$$\bigtriangledown \hat{v}(s, w) = x(s) \tag{2.15}$$

Thus the general SGD update defined in XX can be simplified to
Represent value function by a linear combination of features

$$\hat{v}(S, w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j \tag{2.16}$$

Objective function is The error in a state s is the square of the difference between the approximate value $\hat{v}$(S,w) and the true value v(S,w).

$$J(w) = \mathbb{E}_{\pi}\left[(v_{\pi}(S) - \hat{v}(S,w))^2\right] \tag{2.17}$$

Linear TD(0) is guaranteed to converge to clobal optimum
One disadvantage of the linear method is that it cannot express any relationship between features. For example, it cannot represent that feature *i* is useful only if feature *j* is not present.
Nevertheless, this approach is sufficient enough for our experiment, which will be described in Chapter XXX.
There are different linear methods to represents states as features, such as polynomials, fourier basis, or radial basis functions to name a few. Feature construction depends on a problem you are solving. In the next section, we introduce *Tile Coding* which will be used for our benchmark.

Tile Coding TODO REFERENCE OF THIS METHOD

State set is represented as a continuous two-dimentionala space. If a state is within the space, then the value of the correnspondingfeature is set to be 1 to indicate that the feature is present, while 0 indicates that the feature is absent. This way of representing the feature is called *binary feature*. *Coarse coding* represents a state with which binary features are present within the space. One area is associated with one weight w, and training at a state will affect the weight of all the areas overlapping that state. the approximate value function will be updated within at all states within the union of the areas, and a point that has more overlap will be more affected, as illustrated in Figure XX.
The size and shape of the areas will determine the degree of the generalisation. Large areaas will have more generalisation the change of the weight in that state will affect all other states within the intersection of the spaces.
The degree of overlap within a space will determined the degree of the generalisation.
The shape of the space also affect how it is generalised.
Tile coding is a type of coarse coding. *Tiling* is a partition of state space, and each element of the partition is called a *tile*.
In order to do coarse coding with tile coding, multiple tilings are required, each tiling is offset from one another by a fraction of a tile width.
As illustrated in Figure XXX, when a state occurs, several features with corresponding tiles become active,
Tile coding has computational advantage, since each component of tiling is binary value, XXXX.
a trained state will be generalised to other states if they are within any of the same tiles.
Similar to coarse coding, the size and shape of tiles will determine the degree of approximation.

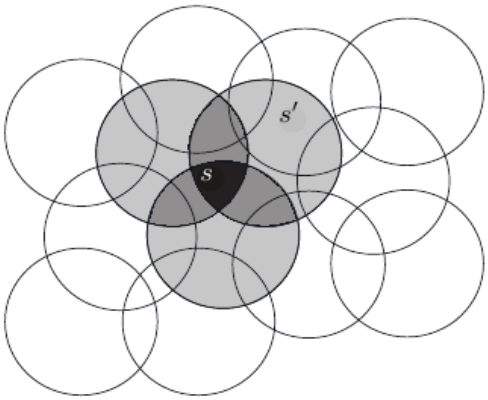

Figure 2.3: Relationships among learning, planning and acting Update this figure

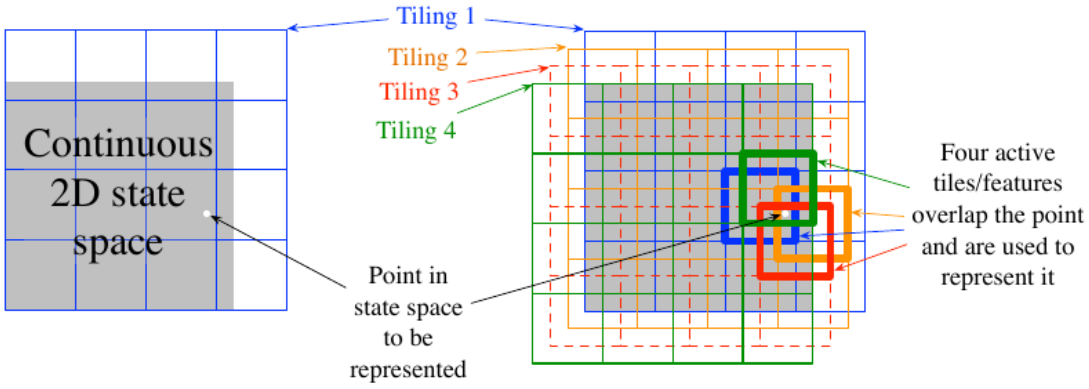

Figure 2.4: Relationships among learning, planning and acting Update this figure

Chapter 3

Implementation and Methodology

3.1 Proposed Architecture

The overall architecture is shown in Figure **??**. By interacting with the environment, the agent accumulates state transition experiences, which is used by ILASP to learn a hypothesis. The agent also record surrounding information it has seen as background knowledge, which will be used to make a plan together with the hypothesis that ILASP learns by solving for answer sets. Each steps are described in details in the following subsections.

3.1.1 Experience Accumulation

Draw an illustration of the difference between exploration and las part
Related these with Definition of ILASP
The first step is to accumulate experience by interacting with the environment. The agent explores the environment randomly until it reaches the goal once. Every time the agent takes an action during the exploration phase, these experiences need to be recorded in two ways: state transition experience and environment experience.

State transition experience

State transition experiences will be used as E in ILASP, especially positive examples for ILASP in ASP syntax, which is of the form

```
#pos({state_after((X2,Y2))},
     {all other state_after that did not happen},
     {state_before((X1,Y1)). action(A). surrounding information}).
```
(3.1)

where,

- inclusions contain one state_after((X2,Y2)), which represents the position of the agent in x and y axis after an action is taken
- exclusions contain all other state_after((X,Y)) that did not occur

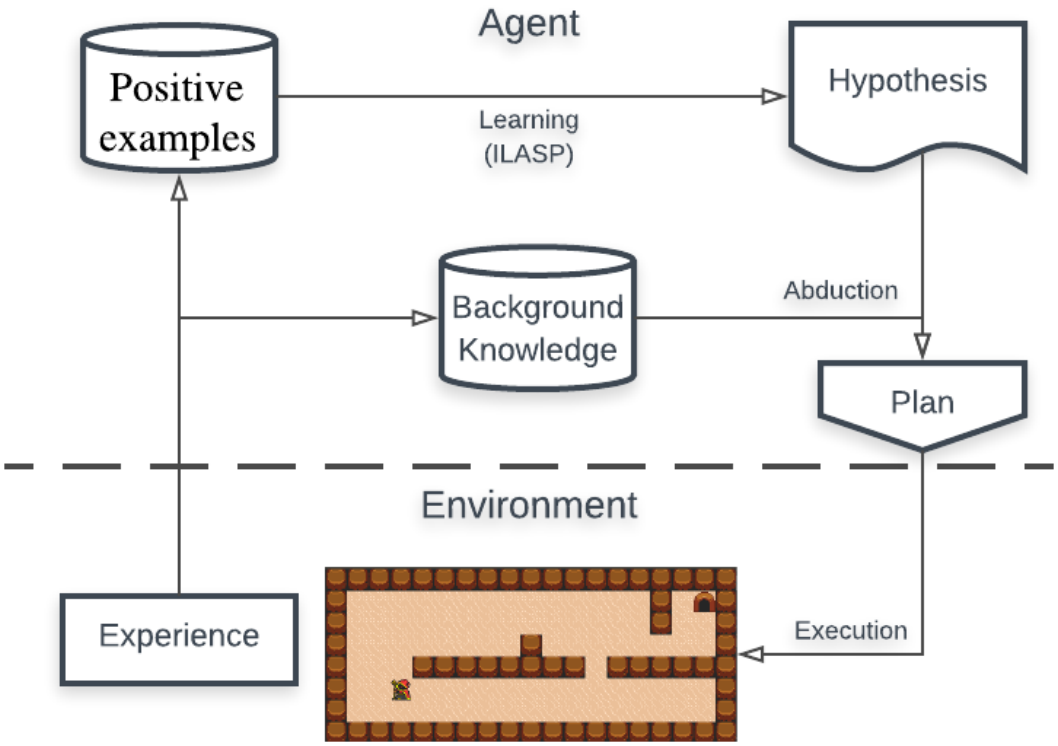

Figure 3.1: Proposed reinforcement learning architecture. ILASP learns to generate a model and updates based on the interaction with the environment, which is used to hypothesis.

- context example include state_before((X1,Y1)), which represents the position of the agent in x and y axis before an action is taken, action(A) is the action the agent has taken, and surrounding information, such as walls, if any.

At the first stage, the input of real experience needs to be converted in ASP form, which can be used to execute the inductive learning in ILASP. The input used in ILASP is state transitions, rewards and an action of the agent, which can be directly converted using a simple mapping table or an action language (such as BC^+ as used in [**?**]). The following ASP input is what is sent to ILASP.

There is no negative example as XXXX.

example 3.1.1. (Positive examples).
Suppose an agent takes an action ”up” to move from (1,3) to (1,4) cell. All other alternative states that the agent could have ended up by taking different actions (down, right, left, and not move) are in the exclusions. Finally surrounding walls information are in the context.

```
#pos({state_after((1,3))},
    {state_after((2,4)),state_after((1,5)),state_after((0,4)),state_after((1,4))},
    {state_before((1,4)). action(up). wall((1, 5)). wall((0, 4)).})
```

This example will be used to learn how to move up as one of the agent's hypotheses. Another example is

```
#pos({state_after((1,3)),
    {state_after((2,3)),state_after((1,4)),state_after((0,3)),state_after((1,2))}
    {state_before((1,3)). action(up). wall((2,3)). wall((0,3)). wall((1,2)).}).
```

where the agent tried to move up, from (1,3) to (1,2), but ended up in the same cell at (1,3). This is because there is a wall at (1,2), and the agent learns in order to move an above cell, there must not be a wall in the above cell.

Environment experience

While the agent explores in the environment, it also remembers all the surrounding information as background knowledge, which will be used to generate a sequence of actions plan using H. In a simple maze, these could be all wall position that the agent has seen so far, which can be
wall((1, 5)). which represents the location of the wall.
Another example could be a location of a teleportation if the agent sees it.
These environment experiences are part of context examples in the positive examples.
Together with the positive examples (E) as described above.

3.1.2 Inductive Learning

Once the agent hits the goal once, ILASP gets triggered and try to learn a hypothesis using the positive examples the agent accumulated so far.
generate hypothesis H,
In addition to the positive examples, the following definitions are supplied

```
cell((0..7, 0..6)).
adjacent(right, (X+1,Y),(X,Y)) :- cell((X,Y)), cell((X+1,Y)).
adjacent(left,(X,Y), (X+1,Y)) :- cell((X,Y)), cell((X+1,Y)).
adjacent(down, (X,Y+1),(X,Y)) :- cell((X,Y)), cell((X,Y+1)).
adjacent(up, (X,Y), (X,Y+1)) :- cell((X,Y)), cell((X,Y+1)).
```
(3.2)

```
#modeh(state_after(var(cell))).
#modeb(1, adjacent(const(action), var(cell), var(cell))).
#modeb(1, state_before(var(cell)), (positive)).
#modeb(1, action(const(action)),(positive)).
#modeb(1, wall(var(cell))).
```
(3.3)

Without these in the form of mode bias, the search space for ILASp will be empty. The full details for ILASP learning tasks is described in Appendix XXX. Positive excludes the possibility of negation as a failure in order to reduce the search space.

```
#max_penalty(100).
```
(3.4)

By default it is XX,

```
#constant(action, right).
#constant(action, left).
#constant(action, down).
#constant(action, up).
```
(3.5)

The actions in the body have to be constant
Together with the above defition as well as accumulated positive examples, ILASP is able to learn an hypothesis. The quality of H depends on the experiences for the agent. For example, In the early phase of learning, the agent does not have many examples, and learns an hypothesis that may not be insightfull. For example, if the agent has only one positive example,
XXX
The learnt hypothesis is XXX
This hypothesis, for example, does not explain how to move ”down”. In order to learn how to move ”down”, it needs an positive example of moving up.
later on H improving as we collect more examples as well as background knowledge.

```
state_after(V0) :- adjacent(right, V0, V1), state_before(V1), action(right), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V0) :- adjacent(down, V0, V1), state_before(V1), action(down), not wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
```
(3.6)

These learnt H will be used to generate a plan in the abduction phase.
After executing the plan, the agent will have more positive examples, which will be used to improve the quality of H.

3.1.3 Generate a plan

Generate a plan using abduction
If the hypotheses were not accurate, clingo might not generate all the actions leading to the goals.

3.1.4 Plan execution

the plan generated by clingo is a set of states and actions.
states are of the form state_at((X,Y),T), where X and Y represent x-axi and y-axi in a maze respectively, T represents a time that the agent is at this particular X,Y cell.
action(A,T) tells which action the agent should take at each time. By following the actions, the agent should collect both predicted state that the agent will end up, and the observed state that the agent actually end up. If there is a difference between these two, either B or H do not correctly represent the model of the true environment, so needs to be improved.
When the agent encounters a new environment (e.g a new wall), this new information will be added to its background, which will be used to improved the hypothesis next time ILASP gets executed.
For example,

$$\begin{aligned}
&\mathsf{state_at}((1,1),1),\ \mathsf{action}(\mathsf{right},1)\\
&\mathsf{state_at}((2,1),2),\ \mathsf{action}(\mathsf{right},2)\\
&\mathsf{state_at}((3,1),3),\ \mathsf{action}(\mathsf{right},3)\\
&\mathsf{state_at}((4,1),4),\ \mathsf{action}(\mathsf{right},4)\\
&\mathsf{state_at}((5,1),5),\cdots
\end{aligned}$$

At the start of the learning, H is usually not correct or too general, using this H will generate lots of answer sets that are not useful for the planning. These examples will be collected and included as exclusions of a new positive example.
For example, XXX
To avoid the agent from being stuck in a sub-optimal plan, the agent deliberately discards the plan and takes an random action with a probability of epsilon (which is less than 1) TODO define this mathematically. When the agent deviates from the planning, it often discovers new information, which will be added to B. Exploration is necessary to make sure that the agent might discovers a shorter path than the current plan, which will be demonstrated in the experiment.
Define them here
Our algorithm works by
It buids the model of the environment by improving two internal concepts: hypothesis H and background knowledge B.
In the further research, we could experiment with a more sophisticated exploration strategy, such as XXX and YYY.
This is formally defined in Algorithm.
Everytime the agent executes an action by following the plan, it checks whether the observed state is that is expected.

Algorithm 4 XXXX

```
1: procedure ILASP(RL) (B AND E)
2:     while True do
3:         H (inductive solutions) ← run ILASP(T)
4:         plan(actions, states) answer sets ← AS(B, H)
5:         while actions in P do
6:             observed state ← run clingo(T)
7:             if observed state ≠ predicted state then
8:                 H ← run ILASP(T)
```

If there is a difference between the two, either
B is incorrect
H is not sophisticated enough,
If that is the case, the agent runs ILASP again using more positive examples it collected during the plan execution.

- States: XXX
- Actions: The agent can move up, down, right or left
- Rewards: XXX
- Transitions: XXX

3.1.5 Exploration

our algorithm kicks in once the agent reaches a goal once. However it is likely that the agent has not seen all the environment and therefore is likely to be in a sub-optimal plan. Therefore, similar to RL algorithm, the agent also has to explore a new state. There are a number of exploration strategy in RL (such as Ask Chris on these papers). One of the most commonly used strategy is ϵ-greedy strategy. As described in Chapter XXX, the agent takes an random action
This strategy may not be appropriate in cases there safety is a priority (since it is random action.) It is simple to implement. In the case of our algorithm, the agent discard the plan from the abduction with a probably of epsilon and takes a random action in order to avoid getting stuck in a sub-optimal path. When the agent takes an random action and move into a new state, the agents creates a new plan from the new state and continue to move forward.
This exploration point will be highlighted in Experiment XXX.
Epsilon needs to be larger than Q-learnig because

Chapter 4

Evaluation

4.0.1 Settings

Having implemented the algorithm, we now describe the experiment configuration and results in this chapter. The two main measurements for the performance of our new architecture are learning efficiency and transfer learning capability as stated in the Introduction. We conducted several different experiments to highlight each aspect of the algorithm.

We use GVG-AI Framework for our experiments, which was created for the General Video Gamea AI Competition[1], a game environment for an agent that should be able to play a wide variety of games without knowing which games are to be played. The underlying language is the Video Game Definition Language (VGDL), which is a high-level description language for 2D video games providing a platform for computational intelligence research ([**?**]). The VGDL allows users to easily craft their own environments, which makes us possible to do various experiments without relying on the default environments. The base game is a maze implemented using VGDL game platform. Example XX was created for the experimet 1 in this report, there are only 3 different types of cells: a goal cell, walls and paths. The agent can take 4 different actions: up, down, right and left.

No dynamic enemies,

The agent receives -1 in any states except the goal state, where it gains a reward of 100.

TODO OPENAI paper citation

The game is formalised as MDP as follows:

I conducted a number of experiments to highlight a different aspects of the algorithms.

To compare the performance of our algorithm, I ran Q-learning, the most commonly used RL algorithm, in the same senarios as a bench mark.

[1] http://www.gvgai.net/

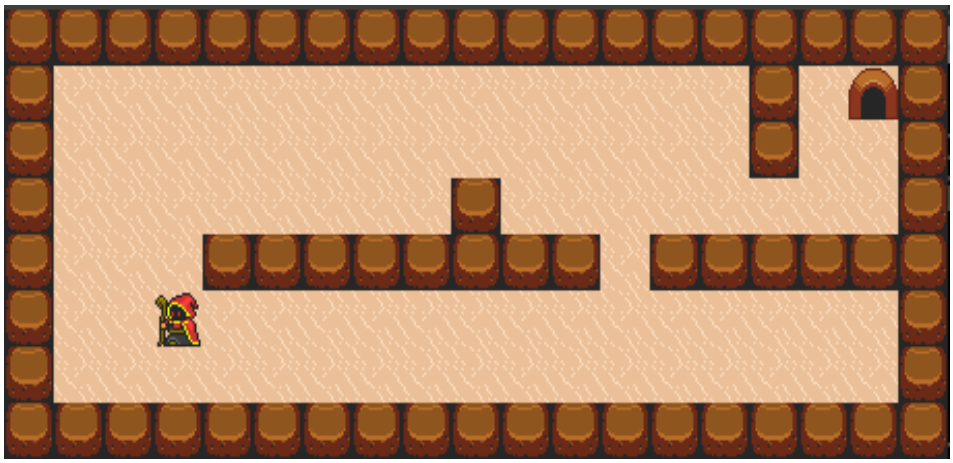

Figure 4.1: VDGL game

4.0.2 Benchmark

We also compare the performance of our algorith with existing reinforcement learning techniques. Q-learning is most widely used technique for reinforcement learning. However comparision with q-learning might not be fair since our algorithm has one extra assumptions: the agent knows surrounding information (whether there are walls in adjacent cells), which is not a common assumption for RL. In order to compare with Another benchmark is tile coding, which is a type of linear function approximation techniques.

4.1 Learning Evaluation

4.1.1 Experiment1

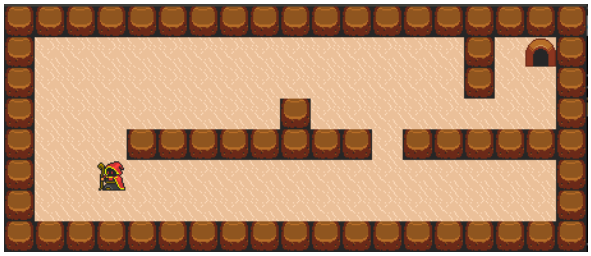

Figure 4.2: GAME

Experiment 1 update H
First H
TODO Insert Hypothesis here
Last H after XX iterations.
TODO Insert Hypothesis here
Converge to optimal policy faster than Q-learning.

Figure 4.3: PLACEHOLDER

4.1.2 Experiment2

The first experiment might not be a fair comparision between our algorithm and Q-learning, since our algorithm has extra information about the surrounding information. In order to have the same assumptions, we use function approximation for Q-learning. Linear function approximation.

Figure 4.4: PLACEHOLDER

4.1.3 Experiment3

Experiment 3 Optimal path learning This experiment is conducted to see if the agent does not get stuck on a suboptimal path.
The game is designed such that
Teleport.

Figure 4.5: PLACEHOLDER

4.2 Transfer Learning Evaluation

4.2.1 Experiment4

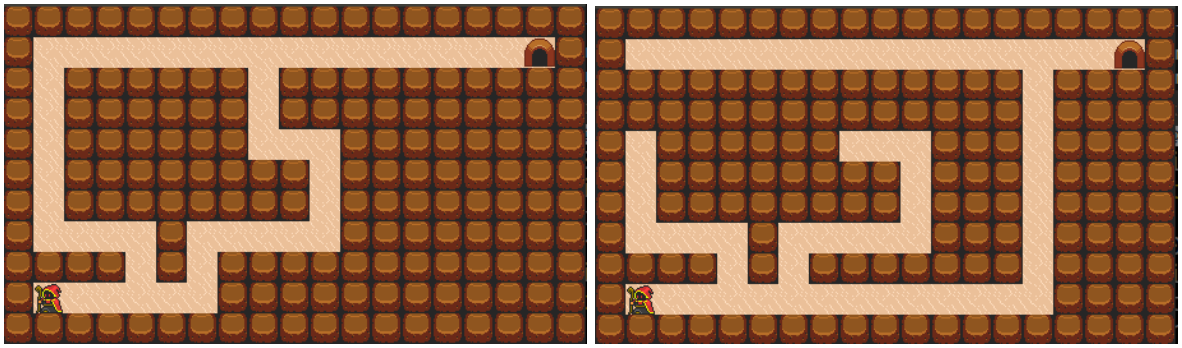

Figure 4.6: Before (left) and after (right) transfer learning

Finally, we investigated the potentials of transfer learning betweeen similar environments.
The goal position is the same as in the first game, but the routes to the goal is different.

Experiment 4 Transfer learning 1 update B 2 update H

Figure 4.7: PLACEHOLDER

4.2.2 Results

4.2.3 Discussion

Strengths

4.2.4 Limitations

You have to define the search space for H
Learning ILASP is known to be less scalable.
ILASP learning is quite slow

Chapter 5

Related Work

In this section, I summarise recent studies related to symbolic (deep) reinforcement learning.
[**?**] introduced Deep Symbolic Reinforcement Learning (DSRL), a proof of concept for incorporating symbolic front end as a means of converting low-dimensional symbolic representation into spatio-temporal representations, which will be the state transitions input of reinforcement learning. DSRL extracts features using convolutional neural newtworks (CNNs) [**?**] and an autoencoder, which are transformed into symbolic representations for relevant object types and positions of the objects. These symbolic representations represent abstract state-space, which are the inputs for the Q-learning algorithm to learn a policy on this particular state-space. DSRL was shown to outperform DRL in stochastic variant environments. However, there are a number of drawbacks to this approach. First, the extraction of the individual objects was done by manually defined threhold of feature activation values, given that the games were geometrically simple. Thus this approach would not scale in geometrically complex games. Second, using deep neural network front-end might also cause a problem. As demonstrated in [**?**], a single irrelevant pixel could dramatically influence the state through the change in CNNs. In addition, while proposed method successfully used symbolic representations to achieve more data-efficient learning, there is still the potential to apply symbolic learning to those symbolic representations to further improve the learning efficiency, which is what we attemp to do in this paper. [**?**] further explored this symbolic abstraction approach by incorporating the relative position of each object with respect to every other object rather than absolute object position. They also assign priority to each Q-value function based on the relative distance of objects from an agent.
[**?**] added relational reinforcement learning, a classical subfield of research aiming to combining reinforcement learning with relational learning or Inductive Logic Programming, which added more abstract planning on top of DSRL approach. The new mode was then applied to much more complicate game environment than that used by [**?**]. This idea of adding planning capability align with our approach of using ILP to improve a RL agent. We explore how to effectively learn the model of the environment and effectively use it to facilitate data-efficient learning and transfer learning capability.
Another approach for using symbolic reinforcement learning is storing heuristics

expressed by knowledge bases [[**?**]). An agent lerans the concept of *Hierarchical Knowledge Bases (HKBs)* (which is defined in more details in [**?**] and [**?**]] at every iteration of training, which contain multiple rules (state-action pairs). The agent then is able to decide itself when it should exploit the heuristic rather than the state-action pairs of the RL using *Strategic Depth*. This approach effectively uses the heuristic knowledge bases, which acts as a sym-symbolic model of the game.
Another field related to our research is the combining of ASP and RL. The original concept of combining ASP and RL was in [**?**], where they developed an algorithm that efficiently finds the optimal solution of an MDP of non-stationary domains by using ASP to find the possible trajectories of an MDP. This approach focused more on efficient update of the Q function rather than inductive learning. In order to find stationary sets, an extension of ASP called BC^+, an action language, was used. BC^+ can directly translate the agent's actions into ASP form, and provide sequences of actions in answer sets.

Chapter 6

Conclusion and Further Research

6.1 Contribution

To my knowledge, this is the first time that both symbolic learning method is incorporated into a reinforcement learning to facilitate learning process

6.2 Further Research

More complicated environment
Dynamic environment
Like moving enermy etc.
Non-stationality possible to be handled??
our approach is similar to experience replay ??

6.2.1 Value iteration approach

The proposed architecture is not finalised and will be reviewed regularly as we do more research. More research needs to be devoted to finalising the overall architecture, and the following issues in particular need to be considered.

6.2.2 Weak Constraint

- Further investigation of whether ILASP can learn the concept of adjacent, which is crucial concept to know in any environment.
- How to generalise the agent's model when the environment changes. The new environment could be very similar to the previous one, or could be a completely different environment thus the agent should create a new internal model rather than generalising the existing model.
- The current proposed architecture is based on Dyna with simulated experiences. However, this might not be the best overall architecture, and the feasi-

bility of using simulated experience with the learnt model with ILASP needs to be further investigated.

- Possibility of using other representational concepts such as *Predictive Representations of State* or *Affordance* [**?**] for the agent's learning task. These concept have not been considered at the moment, but could help better transfer learning.
- Preparation for a backup plan in case ILASP approach does not work, so that the researchs feasible within 3 months of the researcheriod.

6.2.3 probabilistic inductive logic programming

instead of ASP

6.2.4 generalisation of the current approach

Learning the concept of being adjacent

6.3 Conclusion

Chapter 7

Ethics

Appendices

.1 Learning tasks

.2 Ethics checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓

Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?	✓	
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
Section 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?	✓	
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓