Detect, Understand, Act
A Neuro-Symbolic Hierarchical Reinforcement Learning Framework

Ludovico Mitchener · David Tuckey · Matthew Crosby · Alessandra Russo

Received: date / Accepted: date

Abstract
In this paper we introduce Detect, Understand, Act (DUA), a neuro-symbolic reinforcement learning framework. The Detect component is composed of a traditional computer vision object detector and tracker. The Act component houses a set of options, high-level actions enacted by pre-trained deep reinforcement learning (DRL) policies. The Understand component provides a novel answer set programming (ASP) paradigm for symbolically implementing a meta-policy over options and effectively learning it using inductive logic programming (ILP). We evaluate our framework on the Animal-AI (AAI) competition testbed, a set of physical cognitive reasoning problems. Given a set of pre-trained DRL policies, DUA requires only a few examples to learn a meta-policy that allows it to improve the state-of-the-art on multiple of the most challenging categories from the testbed. DUA constitutes the first holistic hybrid integration of computer vision, ILP and DRL applied to an AAI-like environment and sets the foundations for further use of ILP in complex DRL challenges.

1 Introduction

Deep reinforcement learning (DRL) involves the use of neural networks as function approximators in a reinforcement learning (RL) setting (Sutton & Barto, 2018). In recent years, DRL systems have worked well when applied to complex games (Berner et al., 2019; Schrittwieser et al., 2020). However, the extent to which excelling at these video games can be used as a real proxy for intelligence is unclear (Crosby et al., 2020). Current state-of-the-art (SOTA) DRL systems seldom exhibit the most basic of human cognitive faculties such as causal inference, spatial reasoning or generalisation (Garnelo & Shanahan, 2019; Crosby, Beyret, & Halina, 2019). For example, in a recent competition using the Animal-AI (AAI) testbed, the top submissions, based on DRL methods, failed to solve common sense physical reasoning tasks from animal cognition such as object permanence and spatial elimination (Crosby et al., 2020).
Additionally, DRL methods inherit the drawbacks of neural networks including: opacity or non-interpretability, poor generalization to samples outside their training distribution, data inefficiency, and they are purely reactive, i.e. they do not explicitly develop high-level abstractions necessary for causal or analogical reasoning which could be reused across tasks (Garnelo, Arulkumaran, & Shanahan, 2016). To address these shortcomings, which map exactly onto the main strengths of symbolic AI, we propose a novel neuro-symbolic framework that combines the strengths of both DRL and symbolic reasoning and learning using the options framework (Garnelo & Shanahan, 2019; Sutton, Precup, & Singh, 1999).

Our framework, called DUA, is divided into three main components: Detect, Understand and Act. The Detect component extracts an interpretable object representation, in the form of a logic program, from the raw data of the environment using traditional methods from computer vision. The Understand component implements a novel Answer Set Programming (ASP) paradigm to learn a symbolic meta-policy over options using inductive logic programming (ILP). Finally, the Act component uses individually trained DRL agents that implement options. The architecture may be loosely thought of as a two-systems solution (Kahneman, 2011; Booch et al., 2020): the DRL options represent the fast, reactive and non-interpretable facets of intelligence while the symbolic meta-policy learning is the substrate of the slow, logically rational and interpretable side of intelligence.

We evaluate our DUA framework on the AAI 2019 competition testbed and demonstrate several key benefits. Given a set of pre-trained options, we demonstrate few-shot learning by only requiring 7 training examples to learn a general meta-policy which transfers within and between tasks to compete on a testbed of 900 unseen arenas. Training only on those 7 examples, DUA achieves state-of-the-art in 7 testbed categories and above the top-10 average in 4 others, compared to results from the competition. Its modular nature allows it to easily incorporate new options and update or learn new meta-policies to solve completely new types of tasks without having to retrain the whole system. Finally, DUA requires no environment rewards to learn meta-policies, making it particularly adept at extremely sparsely rewarded settings. This work constitutes the first holistic hybrid integration of computer vision, ILP and DRL able to solve common sense physical reasoning tasks such as the animal cognition tasks in the AAI-like environment.

Our contributions are therefore fivefold:

- We propose a novel and general RL algorithm for learning first-order symbolic meta-policies using ILP.
- We present a novel twist on hierarchical reinforcement learning (HRL) to integrate deep and symbolic learning in an RL setting.
- We showcase the first hybrid integration of computer vision, DRL and ILP.
- We adopt a new Event-Calculus inspired ASP paradigm to coordinate such hybrid integration with agency.
- We evaluate our proposed framework in the AAI environment on the full 2019 competition testbed and achieve SOTA in multiple of the most challenging categories.

The paper is organised as follows. In Section 2 we introduce the AAI environment and both the RL and ILP background required for the rest of the paper. In Section 3 we briefly discuss the most relevant approaches in the literature. The rest of the paper describes the DUA framework in detail (Section 4) followed by the
experiments (Section 5) and results (Section 6). We finally conclude and suggest avenues for future work.

2 Background

In this section we introduce the evaluation testbed and a brief summary of the RL and ILP background material used in the paper.

2.1 Animal-AI

The AAI environment (Crosby et al., 2020) comprises of a small arena in which various objects can be placed to recreate tasks used in animal cognition. To simplify the environment in order to focus on the cognitive abilities being tested, the objects are colour coded and of relatively few base types e.g. walls, ramps and food (reward objects). To complete a task successfully an agent has to navigate the environment to collect a predetermined amount of food (reward).

The environment uses the Unity physics engine to simulate realistic physical behaviors such as gravity, friction, acceleration and collisions, and is built on top of ml-agents (Juliani et al., 2018). The virtual equivalent of food is a green sphere with associated reward proportional to its size. The agent also receives a constant, small and negative, reward of \(-1/T\) at every time step, where \(T\) is the maximum number of time steps per episode. The agent’s observations are comprised of coloured pixel inputs of configurable resolution along with a three-dimensional velocity vector. The agent uses a simple discrete action space capable of turning left, right and going forwards or backwards.

The testbed consists of 900 tests broken down into categories, roughly corresponding to different cognitive skills, such as object permanence or causal reasoning. Many categories are incredibly challenging for current SOTA DRL models. For example, the spatial elimination category includes 27 tasks, only 7 of which were solved in the competition. These tasks involve inferring the only possible location that food could be in (behind an opaque object) and directing exploration in that area. These tasks are purposefully designed such that an undirected (e.g. random) exploration strategy will fail. On the other hand, it is not possible to apply symbolic learning methods directly to the environment due to the pixel inputs and low-level control provided by the action space.

Tests within a category in AAI may vary greatly in terms of types of objects encountered and the layout of the environment. For example, the spatial elimination category involves tests with maze-like arenas composed of walls, as well as other tests involving forced-choice tasks with cylinders and blue platforms. As such, the object types, their spatial configuration and the manner in which a cognitive skill is being assessed all vary greatly within a category.

2.2 Reinforcement Learning

Reinforcement learning (Sutton & Barto, 2018) is a general method for training agents to maximise cumulative reward. The problem is usually represented as a
Markov Decision Process, a tuple $M = (S, A, p, r, \gamma)$, where $S$ and $A$ are respectively a finite set of states and actions, $p : S \times A \rightarrow \delta(S)$ is the transition probability function, $r : S \times A \times S \rightarrow \mathbb{R}$ the reward function and finally $\gamma \in [0, 1)$ the discount factor. Initially the optimisation problem may be formulated as, given a state, choose an action that leads to highest expected return. This is known as the action value function $Q^\pi$ for policy $\pi$. The Q-value of a state-action pair may be estimated from experience. By storing the average discounted return for taking an action from each state, the averages will converge to the true action values $Q(s, a)$.

Hierarchical Reinforcement Learning (HRL) leverages the intrinsic compositionality of goals and sub-goals to simplify complex tasks using a divide and conquer strategy. Theoretically, decomposing a problem hierarchically can greatly reduce both space and time complexity in the learning and execution of the overall task (van den Bosch et al., 2011).

Options (Sutton et al., 1999) are one of the most popular formulation of HRL. They allow the RL agent to be divided into three components: primitive actions, temporally extended actions composed of primitive actions, called options, and a high-level policy over options. The high-level policy decides which option to initiate at a given state. Options are executed until a termination criterion is met, usually reaching a sub-goal or a timeout. The high-level policy is then queried again to decide which option should be executed next.

2.3 Inductive Learning of Answer Set Programs

Answer Set Programming (ASP) (Gelfond & Lifschitz, 2000) is a declarative programming paradigm used for knowledge representation and reasoning. We assume a first-order ASP language composed of atoms, of the form $p(t_1, \ldots, t_n)$, where $p$ is a predicate of arity $n$ ($n \geq 0$) and $t_1, \ldots, t_n$ are terms (i.e. constants or variables), and negative atoms, of the form $\neg p(t_1, \ldots, t_n)$ where $\neg$ represents negation as failure (K. Clark, 1987). A literal is an atom or a negative atom. Normal rules are of the form:

$$h : - b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

where $h, b_1, \ldots, b_m, c_1, \ldots, c_n$ are atoms, $n \geq 0$ and $m \geq 0$. We refer to $h$ as the head of the rule and $b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$ (collectively) as the body of the rule. A normal rule with $n = m = 0$ is also referred to as a fact. We assume normal rules to be safe, that is every variable in a rule occurs in at least one positive literal in the body of the rule. A normal rule is ground if it does not contain variables. Given an ASP program $P$, composed of a set of normal rules, the Herbrand Base of $P$ denoted as $HB_P$, is the set of all ground atoms that can be formed from predicates and constants in $P$. An Herbrand interpretation, $I$, is a subset of $HB_P$.

Solutions (i.e. models) of an ASP program $P$ are defined in terms of the reduct of $P$. Given an ASP program $P$, composed of a set of normal rules, and an Herbrand interpretation $I \subseteq HB_P$, the reduct of $P$, denoted as $P^I$, is constructed from the grounding of $P$ by (i) removing all the rules whose bodies contain the negation of an atom in $I$, and (ii) removing all negative atoms from the remaining rules. All rules in the reduct $P^I$ have no negative atoms in the body. An interpretation $I_1 \subseteq HB_P$ is an Herbrand model of the reduct $P^I$ if every rule in $P^I$ is true in $I_1$.

Given a finite set $X, \delta(X) = \{ \mu \in \mathbb{R}^X : \sum_{x \in X} \mu(x) = 1, \mu(x) \geq 0 \}$ is the probability simplex over $X$. 

---

1 Given a finite set $X, \delta(X) = \{ \mu \in \mathbb{R}^X : \sum_{x \in X} \mu(x) = 1, \mu(x) \geq 0 \}$ is the probability simplex over $X$. 

Consider the ASP program given in Example 1, extended with the sets. For further details, see (Calimeri et al., 2019). Throughout the paper we denote the set of answer sets of a program $P$ with $AS(P)$.

**Example 1** Consider the ASP program $P$ given below, and the interpretation $I_1 = \{\text{person}(steve), \text{goDoctor}(steve)\}$. The reduct $P^{I_1}$ is the program $P^{I_1} = \{\text{goDoctor}(steve) :- \text{person}(steve), \text{sick}(steve), \text{sick}(steve) :- \text{person}(steve), \text{healthy}(steve) :- \text{person}(steve), \text{person}(steve)\}$. The reduct $P^{I_1}$ has the minimal model $\{\text{goDoctor}(steve), \text{sick}(steve), \text{healthy}(steve), \text{person}(steve)\}$ which is not equal to $I_1$, so $I_1$ is not an answer set of $P$. The program $P$ has three answer sets, $A_1 = \{\text{healthy}(steve), \text{person}(steve)\}$, $A_2 = \{\text{sick}(steve), \text{call999}(steve), \text{person}(steve)\}$, and $A_3 = \{\text{sick}(steve), \text{goDoctor}(steve), \text{person}(steve)\}$. They intuitively state that solutions to the program $P$ are situations where steve is sick or healthy. If steve is sick, he either goes to the doctor or calls 999. For a more detailed explanation of the semantics of ASP programs, please see (Gelfond & Lifschitz, 2000).

$$P = \begin{cases} 
\text{call999}(X) :- \text{person}(X), \text{sick}(X), \text{not goDoctor}(X). \\
\text{goDoctor}(X) :- \text{person}(X), \text{sick}(X), \text{not call999}(X). \\
\text{sick}(X) :- \text{person}(X), \text{not healthy}(X). \\
\text{healthy}(X) :- \text{person}(X), \text{not sick}(X). \\
\text{person}(steve).
\end{cases}$$

ASP allows also optimisation over the answer sets, according to weak constraints. These are rules of the form:

$$\neg b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_n, [w \oplus p, t_1, \ldots, t_k]$$

where $b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_n$ for (collectively) the body of the constraint, $w$ and $p$ are integers specifying respectively the weight and the priority level, and $t_1, \ldots, t_k$ are terms that appear in the body of the constraint. We refer to $[w \oplus p, t_1, \ldots, t_k]$ as the tail of the constraint. A ground instance of a weak constraint $W$ is obtained by replacing all variables in $W$ (including those in the tail of $W$) with ground terms. Weak constraints do not affect what is, or is not, in an answer set of a program $P$. They create an ordering over the set $AS(P)$ of answer sets, which defines which answer sets are better than another. Informally, given a program $P$ with weak constraints, and an interpretation $I$, we can construct the set of tuples $(w, p, t_1, \ldots, t_k)$ for which there is a ground instance of a weak constraint in $P$ whose body is satisfied in $I$ and whose (ground) tail is $[w \oplus p, t_1, \ldots, t_k]$. At each level $p$, the score of $I$ is given by the sum of the weights $w$ of all such tuples with level $p$. So an interpretation $I_1$ is better than an interpretation $I_2$, written $I_1 \prec I_2$, if there is a level $p$ for which $I_1$ has a lower score than $I_2$ and there is level higher than $p$ for which $I_1$ and $I_2$ have different score. An answer set $A \in AS(P)$ is optimal if there is no other answer set $A_1 \in AS(P)$ that is better than $A$. Note that an ASP program $P$ with weak constraints may have multiple optimal answer sets. For further details, see (Calimeri et al., 2019).

**Example 2** Consider the ASP program given in Example 1, extended with the following weak constraints:
Applying the weak constraints to the three answer sets of \( P \) (given in Example 1), at priority level 1, \( A_1 \) and \( A_2 \) have equal lower score (equal to 0) than \( A_3 \), which has score 5. But at the higher priority level 2, \( A_1 \) has lower score (still equal to 0) than \( A_3 \), which has score 2). So, \( A_1 \) is the optimal answer set, followed by \( A_3 \) and then \( A_2 \).

In this paper, we consider ASP programs to be composed of normal rules and weak constraints.

**ILASP** (Law, Russo, & Broda, 2020) is an ILP framework for learning ASP programs. It includes a family of SOTA systems capable of learning (in principle) any class of ASP program. We present here an adapted definition of the notion of inductive learning of answer set programs that is specific to the class of ASP programs that are learned by our DUA framework\(^2\). The task of learning from answer sets makes use of two types of examples: context-dependent partial interpretations and context-dependent ordering examples. Intuitively, a learned hypothesis (or ASP program) complies with the bias \( S_M \) of context-dependent ordering examples. An ASP program \( P \) bravely respects \( o \) if there is at least one pair of answer sets \( \langle A_1, A_2 \rangle \), where \( A_1 \in AS(P \cup C_1) \) and \( A_2 \in AS(P \cup C_2) \), such that \( A_1 \) extends \( e_1 \), \( A_2 \) extends \( e_2 \) and \( A_1 \prec A_2 \).

In our DUA framework, a learning from answer set task \( T \) is formulated as:

\[
T = \langle B, S_M, E, O \rangle
\]  

where \( B \) is an ASP program called background knowledge, \( S_M \) set of rules (normal rules and weak constraints) allowed in hypotheses, called hypothesis space, \( E \) is a finite set of context-dependent partial interpretations, called examples, and \( O \) is a finite set of context-dependent ordering examples. An hypothesis \( H \) is an inductive solution of \( T \) if and only if the following conditions hold: (i) \( H \subseteq S_M \), (ii) for every \( \langle e, C \rangle \in E \) there exists an answer set \( A \in AS(B \cup C \cup H) \) that extends \( e \); (iii) for every \( o \in O \), \( B \cup H \) bravely respect \( o \). Learning an answer set program \( H \) means computing an inductive solution of a given learning from answer set task \( T = \langle B, S_M, E, O \rangle \). Intuitively, a learned hypothesis (or ASP program) complies with the bias \( S_M \), covers all the given examples and includes weak constraints so that its answer sets respect the given ordering examples. DUA uses the ILASP system (Law et al., 2020) to compute inductive solutions that are essentially meta-policies over options. In Section 4 we describe how a learning from answer set task \( T = \langle B, S_M, E, O \rangle \) is defined to compute such policies, in particular what the hypothesis space \( S_M \) is, and how examples \( E \) and \( O \) are generated.

In DUA we make use of all these methods. The Understand component learns an ASP program with weak constraints that defines an agent’s high-level policy

\(^2\) For a general definition of the learning from Answer Sets framework the reader is referred to (Law, Russo, & Broda, 2018).
over options. The Act component houses the options: low-level policies learned using DRL. Finally, the integration of low-level and high-level policies is inspired by the options framework from HRL.

3 Related Work

In recent years an increasing body of research has been dedicated to merging symbolic and neural systems in an attempt to reap the advantages of both (Marcus, 2020). Such systems have proven their worth on various tasks ranging from reasoning on unstructured data (Minervini, Bošnjak, Rocktäschel, Riedel, & Grefenstette, 2019; Gupta, Lin, Roth, Singh, & Gardner, 2019; Cunnington, Russo, Law, Lobo, & Kaplan, 2020), to visual question answering (Mao, Gan, Kohli, Tenenbaum, & Wu, 2019; Yi et al., n.d.; Han et al., n.d.), to learning proofs (Fawzi, Malinowski, Fawzi, & Fawzi, 2019; Cranmer, Xu, Battaglia, & Ho, 2019), to competing in RL tasks (Zamani, Magg, Weber, & Wermter, 2017; Bougie, Kai Cheng, & Ichise, 2018; Garnelo et al., 2016) and even solving 8th grade science exams (P. Clark et al., 2019). Neuro-symbolic methods may be broadly separated into those that attempt to fuse symbols into the fabric of neural networks themselves (Dong et al., 2019; Liao & Poggio, 2017; Zhang & Sornette, 2017; d’Avila Garcez et al., 2019; Manhaeve et al., 2018) and those that connect the two by either using neural networks to bring unstructured data amenable to symbolic systems or enhance deep systems with symbolic priors. Our approach falls within the latter and so will our overview of related work, in particular within RL.

Garnelo et al. (2016) were amongst the first to show the promise of hybrid methods in RL. Using symbolic common-sense priors, such as object permanence, the authors augment their observation space for a simple RL task. They show that their method generalizes better than a simple DQN to unseen, similar tasks.

More recently, others have followed suit (Zamani et al., 2017; Bougie et al., 2018) in augmenting observation spaces with symbolic representations of their environments to give their agents strong informative priors. Zamani et al. (2017) use a symbolic representation composed of subgoals that boost RL performance by providing intermediate rewards. The work from Bougie et al. (2018) is more directly related to our approach as it is also tested in a complex partially-observable video game environment. They employ a similar pipeline approach whereby the agent receives images as input and the images are enhanced by adding strong symbolic priors related to the environment. Both Zamani et al. (2017) and Bougie et al. (2018) demonstrate significant improvements in results over their purely DRL counterparts.

Another interesting approach comes from Furelos-Blanco et al. (2021). Induction subgoal automata (ISA) uses ASP within the context of HRL, not only to learn the hierarchical structure of the automata, but also the sub-policies themselves. ISA is fully interpretable and trained in a non-differentiable, yet end-to-end fashion. Although its implementation is purely symbolic, the authors suggest ways in which it could use DQN rather than tabular-Q learning. The symbolic inference and induction of hierarchical options in ISA shares similarities with our own approach, namely the use of the HRL options framework, ASP and ILASP. Other approaches also based on a similar idea of using or learning reward automata to guide the RL agent include (Hasanbeig, Jeppu, Abate, Melham, & Kroening, 2019;
Xu et al., 2020), where reward automata are inferred, by SAT solving, from exploration traces and used to “orchestrate” sequencing of low-level actions in the RL agent, and (Icarte, Klassen, Valenzano, & McIlraith, 2018) where reward automata are manually engineered and used in an interleaved fashion with the RL agent’s exploration. These existing reward automaton based methods differ from DUA in the fact that our meta-policy is learned from execution traces and not inferred using SAT solving or manually engineered, and it is not used to compute at each iteration, but mainly to guide the choice of options.

Neuro-symbolic techniques have also been used to efficiently verify the safety of DRL policies for use cases where safety violations are unacceptable (Anderson, Verma, Dillig, & Chaudhuri, 2020). Relational reasoning inspired by symbolic AI has also been shown to be beneficial in certain RL environments (Shanahan et al., 2020). Furthermore, with the growing importance of graph theory within ML, graphs are increasingly being used to represent compositional scene structure and symbolic relations (Jiang, Dun, Huang, & Lu, 2018; Hart & Knoll, 2020). Graph neural networks are number and order invariant, while explicitly incorporating relations, voiding the need for them to be inferred. This makes graphs ideal candidates for semantic environments, benefiting from object-centric understanding (Hart & Knoll, 2020).

Others have explored the use of program synthesis applied to RL. In Sun et al. (2020) and Andreas et al. (2017) high-level policies are hard-coded and then the options are learned using RL. While the idea of using a program for high-level policies is similar to our approach, our approach differs in the following ways. We learn both the high-level policy and the options, albeit separately with the options being pre-trained. We use inductive learning of ASP programs that supports relational knowledge discovery rather than function-based program induction and does not rely on types. Our learning components can learn programs with general relations, using also non-monotonic semantics in the presence of incomplete information, which is not applicable to program synthesis. As such, program synthesis can be seen as a special case of our symbolic learning approach in which general relations are restricted to functional relations. A final distinction is that in our DUA framework the symbolic system is the “reinforcement learner” (i.e. learning the policy in the shape of weak constraints) and not used to guide a separate RL model as in other works of program synthesis (Yang et al., 2021) and generalised planning (Srivastava, 2011; Icarte et al., 2018).

These works provide a promising glimpse into what is possible by boosting DRL methods with meaningful symbolic-informed priors. Not only do they often increase performance and data-efficiency, but they also allow for a higher degree of interpretability. To the best of our knowledge, however, there has been no neuro-symbolic RL method that goes beyond using symbolic AI simply as an inductive bias rather than as a central component in a complex 3D environment, as it is the case of our DUA framework. No example has been found of neuro-symbolic RL agents that benefit from the expressivity and symbolic dexterity afforded by formal logic programming (LP) languages such as ASP, and methods for learning ASP programs such as the ILASP system used in DUA.
Fig. 1: Example macro-step through the Detect, Understand, Act architecture.

4 DUA

We now introduce the DUA approach. First with a high-level overview, and then with a detailed description of each of its components. Although DUA is a general framework, to illustrate its components better, we describe how each component has been designed to learn and solve tasks in the AAI environment. In Section 6.3 we comment on applying the DUA approach beyond the AAI environment.

4.1 Overview

DUA operates on two different levels of temporal abstraction. The lower level operates in the same time and action space as the RL environment. This level of temporal abstraction will be referred to as the *micro-level*. DUA is capable of initiating actions referred to as *options* which persist across often hundreds of environment timesteps. This timescale will be referred to as the *macro-level*. DUA has two types of policies, a high-level meta-policy on the *macro-level* that maps symbolic states to options, and the options themselves which map environment observations to discrete actions on the *micro-level*.

DUA is named after its three components: Detect, Understand and Act (see Figure 1 for instantiation of DUA in the AAI environment). The Detect module receives information from the environment at each timestep and filters it into a meaningful representation. The Understand (reasoning and learning) module processes this symbolic representation of the environment and infers the correct option to initiate given the current state by using the learned meta-policy. The Act component is composed of the options which are pre-trained DRL agents. Each option takes as input a filtered version of environment observations based on the instructions of the Understand component. For example, if the Understand component decides to ‘interact with object x’, only features of the environment pertaining to object x will be fed to the corresponding option. The option will then execute until a stopping criterion is met and a new query to the Understand component is made to decide on the next option to execute.

4.2 Detect

This module serves to “ground the world.” For an agent in an RL environment, the role of the Detect module is to filter the raw and noisy image tensor into the salient features which are most useful to maximise its reward. The Detect module
parses the image into a set of bounding boxes, making use of the colour coding of objects used in AAI. The object detector recognises colour ranges and associates them with known object types. We use centroid tracking to keep track of objects over time (Nascimento, Abrantes, & Marques, 1999). Objects no longer visible persist in memory for a preset number of timesteps.

Finally, the Detect component translates the bounding boxes’ information into an ASP program composed of ground facts. It also computes simple arithmetic-based heuristics over bounding boxes to detect relations between objects in the scene, such as relative position, and adds it to the ASP program. We call this set of facts in the generated ASP program the observables. An example of facts generated is the following:

\[
\text{platform}(1). \\
\text{goal}(0). \\
\text{on}(0, 1).
\]

stating that a goal on a platform is visible to the agent. The numbers are identifiers given by the centroid tracking to each object. The calculations used to determine \text{on}(x, y) and other relations are detailed in the Appendix.

4.3 Understand

The Understand module may be considered to be the foundation of our approach. It is in charge of learning how to act appropriately to solve tasks, and reasoning over the high-level symbolic state of the environment. The Understand component is itself split into two sub-components: 1) an ASP program containing the meta-policy (policy over options) and common sense background knowledge (detailed in the Appendix) and 2) the ILASP learner which learns the meta-policy.

When queried, the Understand module adds to its ASP program the set of observables and outputs the optimal option to execute. The ASP program contains a set of rules that augment the observation space from the Detect module with common sense rules, such as a goal is always present even if not visible, and a set of rules that instantiate all of the possible options to execute, together with what type of object they should attend to. There is one option per answer set of this ASP program: the answer sets of this program represent all of the possible options that can be selected at a given time. The meta-policy itself takes the form of a set of weak constraints that rank the answer sets and thus the possible options. The option to execute is the one corresponding to the optimal answer set (when multiple answer sets are optimal, one is chosen at random). The set of weak constraints are learned from environment traces as described in Section 4.5.

The representation of sequential events in our ASP program draws inspiration from Event-Calculus (Sadri & Kowalski, 1995; Kowalski & Sergot, 1989). Time is decomposed into discrete events over which our program reasons and decides what options to execute following certain events. Although the events themselves span over irregular time frames in the environment, the events are perceived as quasi-instantaneous by the ASP program which reasons over a single event at a time.
4.4 Act

The Act component houses the set of options, which are pre-trained DRL agents that correspond to sub-goals. In our application to AAI, we use 9 pre-trained options (detailed in the Appendix). These are:

- \texttt{interact}(X): goes to touch object X
- \texttt{explore}(X,Y): explores behind object X to find object Y
- \texttt{balance}(X,Y): traverses along object X without falling to reach object Y
- \texttt{climb}(X): climbs up object X (for ramps)

The Act component receives the identifier of the option to execute, along with some configuration indicating the stopping criteria and what its inputs are. For example, when we climb an object with identifier X, the bounding box of the object X is fed as input to the climb policy which terminates when the agent has reached the peak of the ramp or times out. It oversees the course of the option in the environment and then calls the Understand module upon termination.

We use Proximal Policy Optimisation (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) as our DRL algorithm of choice for AAI as it works with discrete action spaces, is easy to implement, requires little fine-tuning, and has been shown to perform well over a wide variety of benchmarks (Schulman et al., 2017).

4.5 Inductive Meta-Policy Learning

This section describes the core of our contribution, that is our approach for learning a symbolic meta-policy over options which we call Inductive Meta-Policy learning (IMP). We collect meta-traces from option-environment interactions and translate them into a learning from answer sets task. These traces are not the environment traces, but the sequence of states and actions as viewed from the macro-level in the Understand module: the state of the world (expressed in the ASP program), when it was queried and which option was then executed. The environment timesteps are ignored in these meta-traces as we are only interested in learning which option to choose, since the execution of such option is left to the Act module.

We formalise the collection of meta-traces as a set $T$ of tuples $\langle G, P \rangle$, where $G$ is a meta-trace and $P$ is a boolean. Each tuple in $T$ corresponds to a collected episode. A meta-trace $G$ is composed of pairs of partially observable symbolic meta-states $s$ and options $o$. A meta-state is composed of all detected observables at a single macro-step, along with all the high-level relations between the agent and the objects inferred (via the background knowledge in the ASP program) by the Understand module. In other words, a meta-state is the set of all the true logical atoms in the Understand module at a given macro-state (when the Understand module is queried). The meta-trace is then the sequence of “symbolic” meta-states of the system and the options executed after each of these states is observed. For simplicity, we shall henceforth refer to meta-states simply as states. The boolean $P$ for each episode represents the success or failure of the episode: -1 means the agent failed to solve the task and 1 means it succeeded. $n$ is the number of meta-traces. Note that importantly, IMP, unlike RL methods, does not use environment reward. Instead, it only considers the binary outcome $P$: whether the meta-trace leads to success or failure.
In order to learn a meta-policy, we need to transform this set $T$ into a learning from answer sets task. Meta-policy learning happens in three steps:

1. Collect the meta-traces by running the agent in the environment and at each macro-step randomly picking options to execute. We store the meta-traces along with their respective episode success in the set of tuples $T$.

2. We abstract each meta-trace: we map the state-option pairs in the meta-traces in $T$ to a set $T_a$ of tuples including the abstract state-option pairs and associated expected return. This step finds in $T$ similar state-option pairs and combines them to obtain a value akin to a Q-value.

3. We map the generated set $T_a$ into a learning from answer set task $T_i$ to learn the meta-policy $\pi_{meta}$.

**Preprocessing: Abstraction and Q-value calculation.** A difficulty arises when attempting to compare two similar symbolic states. Take for example the two state-option pairs:

- $\text{goal}(0), \text{platform}(1), \text{on}(0, 1), \text{interact}(0)$
- $\text{goal}(1), \text{platform}(2), \text{on}(1, 2), \text{interact}(1)$

They are equivalent, yet they differ due to the id assigned by the centroid tracking. To abstract away from object identifiers we modify the atoms such that the specific identifiers are replaced with abstract tokens. For example, the symbolic state-option pairs in the previous example both become $\text{goal}(a), \text{platform}(b), \text{on}(a, b), \text{interact}(a)$. This allows us to recognise that multiple state-option pairs correspond to the same abstract state-option pair and are thus comparable. We shall refer to these abstract state-option pairs henceforth as abstract pairs.

Now, to obtain a numerical value akin to a Q-value, we assign to the last state in each meta-trace $G$ a reward of 1 or $-1$, given by the value of $P$ associated with $G$ in the tuple $\langle G, P \rangle \in T$. All preceding state-option pairs in the same meta-trace are then assigned a discounted return using a discount factor $\gamma$, as it is common in reinforcement learning. Since the state-option pairs are merged into their respective abstract pairs, we average their associated reward to compute the expected return, or Q-value, for each abstract pair:

$$Q(\bar{s}, \bar{o}) = \frac{1}{k} \sum_{i=1}^{k} R_i$$

where $\bar{s}$ is an abstract state, $R_i$ are all individual discounted rewards associated with state-option pairs that are equivalent to the abstract pair $\{\bar{s}, \bar{o}\}$ and $k$ is the number of such equivalent state-option pairs.

The two pre-processing steps (i.e. computation of abstract pairs and calculation of associated Q-values), produce at the end a set $T_a$ of tuples $\langle \bar{s}, \bar{o}, Q(\bar{s}, \bar{o}) \rangle$, containing all abstract pairs and their associated expected return. This set $T_a$ is used to generate a learning from answer set task $T_i = \langle B, S_M, E, O \rangle$ as defined below. A solution $H$ to this learning task is a set of weak constraints that we call a meta-policy.

**Constructing the learning from answer sets task.** The Understand component of DUA generates a learning from answer set task $T_i = \langle B, S_M, E, O \rangle$. The **background knowledge** $B = \emptyset$, the **hypothesis space** $S_M$ is defined as set of weak constraints of the form

$$\vdash \bar{o}, \text{ob}_1, \ldots, \text{ob}_k[-1@l, M]$$
where \( \bar{o} \) is a single positive option, \( \bar{ob}_1, \ldots, \bar{ob}_i \), for \( i \leq n \) are (negative) observables, and \( n \) is the maximum number of literals allowed in a rule. The tail \([-101, M]\) of each weak constraint has weight \(-1\), a priority level 1 and as \( M \), all the variable terms that appear in the body of the constraint. Each of these weak constraints represents a preference to execute the option \( \bar{o} \) if the condition described by \( \bar{ob}_1 \ldots \bar{ob}_i \) is met. The maximum priority level allowed is equal to 1.5 times the number of options. This ensures that we have a priority level for every option as well as some margin for capturing more complex dependencies. Object types such as \textit{wall}(X) are not included in the hypothesis space as they are implicit in the construction of the option space.

The examples \( E \) is a set of pairs \( e_i = \langle \langle e_{inc}^i, e_{exc}^i \rangle, c_i \rangle \), each representing an abstract pair. The partial interpretation \( \langle e_{inc}^i, e_{exc}^i \rangle \) of each example is empty, i.e. \( e_{inc}^i = e_{exc}^i = \emptyset \), and the context \( c_i \) of each example \( e_i \) is an abstract pair represented as a set of facts\(^3\). The ordering examples \( O \) define ordering pairs over the examples in \( E \). It is this set \( O \) of ordering examples what allows us to express preference over choosing an option over another for a given state. For every abstract state the single optimal option, that is the option with highest expected return in \( T_a \), is pairwise ordered with respect to all other options taken from that abstract state. In other words, we ask ILASP to prefer the answer sets where this optimal option appears for its given abstract state. There are no orderings between abstract states nor between sub-optimal options within abstract states. For example, in the abstract state where a goal and two walls are visible, interact with goal is preferred to rotate and avoid goal. However, there is no ordering between rotate and avoid goal. This would be represented, for instance, by the following context-dependent examples \( e_1, e_2 \) and \( e_3 \) and related ordering examples \( o_1 \) and \( o_2 \):

\[
\begin{align*}
\text{e}_1 &= \langle \langle \emptyset, \emptyset \rangle, \{\text{goal}(a), \text{initiate}(\text{interact}(a)), \text{wall}(b), \text{wall}(c)\} \rangle \\
\text{e}_2 &= \langle \langle \emptyset, \emptyset \rangle, \{\text{goal}(a), \text{initiate}(\text{rotate}, \text{wall}(b), \text{wall}(c))\} \rangle \\
\text{e}_3 &= \langle \langle \emptyset, \emptyset \rangle, \{\text{goal}(a), \text{initiate}(\text{avoid}(a), \text{wall}(b), \text{wall}(c))\} \rangle \\
\text{o}_1 &= \langle \text{e}_1 \prec \text{e}_2 \rangle \\
\text{o}_2 &= \langle \text{e}_1 \prec \text{e}_3 \rangle
\end{align*}
\]

5 Experimental Setup

5.1 Option Training

Each option is trained with identical hyperparameters i.e. no hyperparameter tuning is necessary. For each option, a distribution of arenas is defined and the agent is trained by randomly drawing an arena from this distribution for each new episode. For example, for the balance option, the arena distribution contained various configurations of goals on platforms, requiring the agent to balance along the platform to reach the goal. The full list of options and their corresponding training environments are described in the Appendix.

\(^3\) Note that the representation of abstract pairs as context of an example is sufficient in our case, since the learning from answer sets task in IMP is aimed at learning only weak constraints.
To accelerate training, we make use of reward shaping as well as observation filtering unique to each option. For example, the balance agent only ‘sees’ the bounding box of the goal and a masked image only showing platforms.

The set of options used is chosen a priori, but not how they are used. The meta-policy learned is dependent on the set of options available. We conducted an experiment analysing how DUA behaves when using different subsets of our set of options. It showed that the framework adapted to the set of options available without having to “know” what the effects of the options are i.e. it can be applied for any set of options.

5.2 IMP Training

Once all the options are pre-trained, we create a training set of 7 arenas (detailed in the Appendix), deemed sufficient for the agent to learn an effective meta-policy. At each macro-step, the agent randomly chooses from the options available. The options are object-type sensitive and so for a given state, only certain options will be available, those for which their object-type is present. Using the example of balance again, this option will only be available when there are platforms detected in the environment. Furthermore, for each training arena, we enforce early stopping by constraining the number of macro-steps to be the minimal number of steps necessary to successfully complete the arena. Training continues on each arena until it has reached a heuristic number of successful episodes. The collection of traces is parallelisable both within and between training arenas as generating traces is independent from one arena to another.

Once the traces are collected, they are pre-processed and learning from answer sets task is automatically generated as described in Section 4.5. The learning system ILASP used by the Understand component returns then a hypothesis. This is stored and used to solve the AAI testbed.

6 Results

In this section we compare the performance of our DUA framework to the submissions to the 2019 AAI competition, and analyse various aspects of inductive meta-policy learning.

6.1 AAI Competition

To evaluate DUA we implemented 9 options and created 7 training arenas. The final meta-policy learned is displayed below:
The above meta-policy may be read as a ranking over options constrained by certain relations. Below is a line by line translation in plain English. A given line is only used if there are no lines above it that are true.

- If a ramp is available then climb it.
- If the agent is on a platform and there is lava near the goal, then observe the arena dynamics.
- If there are more goals on one side of a platform you are on, then go to that side.
- If there’s no lava, collect multi-goals.
- If there’s no lava around the goal and the goal is not on a platform go get it.
- Explore the object most likely to be occluding the goal.
- If an object occludes the goal, go explore it.
- If there is lava, fetch the goal while avoiding lava.
- If the agent is on the platform, then balance on the platform to get to goal.
- If goal V1 is bigger than goal V2, go get goal V2.
- If nothing is visible, rotate 360 degrees until an object is visible.

As such, the final policy can be analysed to give insights into reasons for the behaviour of the agent and is therefore interpretable to some extent.

Figure 2a shows our method compared to the current high score within each category achieved by any of the 60 submissions submitted to the 2019 competition.

We are comparing our model against the best of all submissions for each individual category. We outperform all 60 competitors’ submissions on multiple of the most challenging categories. DUA achieves state-of-the-art results in all the categories related to the 7 training arenas, with the exception of y-mazes, where it still outperforms the top 10 average. This suggests that the meta-policy learned is robust and can generalise to a variety of cognitive reasoning tasks outside its training distribution.

The Overall scores reported in the results include even the tests for which the agent cannot detect the objects in the domain. Due to limitations of the Detect module using colour references, there are many object types (e.g. transparent walls and boxes) that are not detected. This means that the agent fails all such tasks. Despite this, it still would have come 3rd overall in the competition.

6.2 Inductive Meta-Policy Learning

We now analyze various aspects of our inductive meta-policy learning (IMP) algorithm including the scalability of IMP at solving new problems by giving it
Fig. 2: (a) Radar plot comparing success rate per category between SOTA, average of top 10 2019 submission and our approach: DUA. (b) Category and overall performance by incremental training set. The training arenas and options are added cumulatively. For example, the blue test run Object Permanence has also been trained on all the previous training arenas from Basic to Spatial Elimination. (c) Category and overall performance by number of successes per training arena observed during meta-policy learning. Scores from all figures represent mean ± s.d. from 10 separate evaluations.
more options, the very small sample of training arenas sufficient to learn a general meta-policy, and finally its convergence properties.

6.2.1 Transfer, Scalability and Generalisation

Unlike current DRL systems which usually require complete retraining to solve tasks outside their training distribution, it is sufficient to provide DUA with a single example of a new task and any options it may require. AAI contains a wide variety of tests for each category, yet we find that DUA only requires one example arena per category in order to generate the results in Figure 2.

To illustrate IMP’s capacity at few-shot generalisation, we analyse the effect of incrementally adding one arena at a time to the training set. In Figure 2b we show how the scores improve as new training arenas and options are added. For example, the first system in red is just trained on the basic food and obstacles arena and does not have any of the options required to avoid red objects or climb ramps. Once we provide it with the avoid option and a single example of a training arena with a red object, the meta-policy adapts to include avoiding red objects and remains robust as more options are added. This is shown in Figure 2b by the jump in performance between the Basic bar and Lava bar on Avoid Red tasks. Likewise for ramp usage, numerosity and object permanence. The scores for some tasks fluctuate because objects in AAI do not always have the same uses. For example, our agent might learn in the ramp usage category to balance on platforms, while in spatial elimination it needs to choose a side of the platform to drop off of. As such, the performance does not necessarily always increase for each category when adding new training arenas and options. A detailed explanation for the results in Figure 2b is included in the Appendix.

6.2.2 Fine-tuning the Meta-Policy

We also investigated how many successful meta-traces IMP requires to learn a general meta-policy effective on the whole testbed. Our results in Figure 2c show that with very few positive examples the overall meta-policy already becomes competent at a wide variety of tasks. However, the competency seems unstable for certain categories such as numerosity and spatial elimination. We interpret this as simple policies are very quickly learned enabling an immediate jump in performance. However, more intricate dependencies require more fine-tuning. For example, when the agent is on a platform, balancing on it towards the goal is only an optimal action if the goal is also on the platform. With the run number 20, the meta-policy misses this and always balances on platforms, leading to losses in the numerosity and y-mazes tasks where the agent must choose a side to drop from the platform.

This quick gain in performance followed by fine-tuning instability is corroborated by analysing the evolution of Q values during training where optimal actions quickly separate from sub-optimal actions, but optimal actions then require further meta-traces to stabilise on slight preferences between optimal actions. In the next section we also analyse the number of successful traces required to learn a meta-policy for each single training arena and again found that only 1-4 examples are required.
6.2.3 One-shot and few-shot learning on individual training arenas

To illustrate the fact that IMP only needs a few successful meta-traces, we trained DUA on one training arena at a time and recorded how many successful meta-traces IMP needed to learn a policy on each. This can be seen in Figure 3. The score is obtained by testing the learned meta-policy on the set of arenas from AAI corresponding to the skills taught by this training arena. In most cases it demonstrates one-shot learning, requiring a single positive example to learn the optimal meta-policy for a single training arena. At the same time, Figure 3 shows that the meta-policy learned on each individual training arena generalises well to the arenas of the same category in the AAI testbed. Interestingly, the higher the likelihood of success from taking random options on a given arena, the more iterations IMP requires to converge. This is to be expected as in categories such as numerosity or y-mazes, often offering the agent a forced choice, choosing randomly usually guarantees success 50% of the time. It is thus harder for the meta-policy learner to dissociate effective vs lucky actions. The number of successes required, however, still remains under four for all arenas from the training set. DUA’s capacity to perform one-shot learning is due to it learning a robust behavioral policy consistent with a high-level interpretable understanding of what cognitive skills the category requires.

6.2.4 Convergence of Meta-Policy Q-values

We also analyse how the distribution of Q-values evolves during meta-policy training. Initially the Q-values are spread out and then very quickly two clusters start to form: a large cluster of sub-optimal abstract pairs with low Q-values and then a smaller cluster of optimal abstract pairs. The final distribution is visualised in Figure 4.
Fig. 4: Final distribution of Q-values at the end of meta-policy training.

Fig. 5: Evolution of number of *abstract pairs* throughout meta-policy learning.

The number of *abstract pairs* experienced following a purely random policy converges to around 120. This indicates that during meta-policy training we have traversed the full search-space multiple times for each *abstract pair*. As such, this justifies our use in this implementation for AAI of one-step learning rather than incrementally updating the meta-policy. Should IMP be applied to larger search spaces, incremental learning would be useful.
6.3 Applying DUA and IMP to new environments

We have proposed a general method for learning and enacting intelligent behaviour in virtual RL environments. DUA contains the scaffolding to interface computer vision, neural actors and symbolic reasoner in a closed loop while IMP symbolically learns a high-level policy over options.

The framework may be applied to any typical RL environment. For each new environment, one needs to decide what are the observables to be used in the ASP representation, choose and train the options and finally implement a detector that translates the input from the environment into observables. It is worth noting that this framework works with any type of detector as this does not influence the shape of the logic program. The core of our framework (learning a symbolic meta-policy) adapts to any environment.

Training the set of options should require no or very little hyper-parameter tuning as each option focuses on learning one skill. In the AAI case, training all options was more than three orders of magnitude faster than other top submissions based on DRL methods which additionally require a considerable amount of hyper-parameter tuning. In this paper we only learn weak constraints that constitute our meta-policy. However, the ILASP system used by our DUA architecture is capable of learning any ASP program. For example, in this work we have encoded in the ASP reasoner the default assumption that "if an object is visible, then it is also present". Such assumptions could also be directly learned using ILASP. As such, this initial framework opens up the opportunity of learning more complex symbolic representations overlaid over deep neural enactors.

7 Conclusions & Future Work

In this paper we have presented a novel neuro-symbolic hierarchical reinforcement learning approach that outperforms previous approaches on challenging physical cognitive tasks. DUA acts effectively in continuous, noisy and high-density domains while maintaining a simplified and goal-driven high-level representation of the environment and its actions. It is capable of identifying objects, their properties and their relations, making consistent decisions on their relevance for solving tasks and finally reporting these inferences in an interpretable manner. We further present a novel algorithm, inductive meta-policy learning, capable of learning from very few examples, which high-level actions to take, given a symbolic representation of the world in extremely sparsely rewarded environments. We discuss the modular quality of our approach, which allows for straightforward generalisation and transfer to further complex tasks.

We have provided evidence that neuro-symbolic approaches can help to solve cognitive tasks. In the future, this can be improved by shifting away from hand-crafted object detectors, allowing for more resilient and accurate object representations. Further systems would ideally learn what options are needed and find a way to leverage intra-option dependencies. Although we used IMP for a single policy update, the system can also be used incrementally. This setup would benefit tasks with larger search spaces of symbolic state-option pairs. Finally, the symbolic dexterity afforded by ASP and ILASP can be further utilised to incorporate more elaborate reasoning on the objects or even on the previous options chosen.
8 Declarations

8.0.1 Funding

No funding was received to assist with the preparation of this manuscript.

8.0.2 Conflicts of interest/Competing interests

The authors have no relevant financial or non-financial interests to disclose.

8.0.3 Availability of data and material

The full Animal-AI environment as well as the 2019 competition testbed and results is publicly available at http://animalaiolympics.com/AAI/.

8.0.4 Code availability

The full code will not be made available. However, the sections relating to Inductive Meta-Policy Learning—our primary contribution—will be made available as supporting material to the manuscript.

8.0.5 Authors’ contributions

All authors contributed to the study conception and design. Material preparation, data collection, coding and analysis were performed by Ludovico Mitchener. The first draft of the manuscript was written by Ludovico Mitchener and all authors edited and commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References


Booch, G., Fabiano, F., Horesh, L., Kate, K., Lenchner, J., Linck, N., ... Srivastava, B. (2020). Thinking fast and slow in ai.


Hasanbeig, M., Jeppu, N. Y., Abate, A., Melham, T., & Kroening, D. (2019). Deepsynth: Program synthesis for automatic task segmentation in deep reinforce-


---

**Appendices**

**A Logic Programming**

For an example of a full IMP ILASP learning task, see imp.lp. For an example of the ASP program used for DUA at inference time see inference.lp. For an example of the ASP program used to generate valid random options during imp training see valid_options.lp.

**B The Options**

We trained 9 options for DUA based on the kind of actions that we expect to be useful in the Animal-AI environment. These are Observe, Rotate, Drop(side), In-
teract(x), Explore(x,y), Collect, Avoid(x,y), Balance(x,y), and Climb(x,y). We will go through each of them, describing their purpose, their inputs and how they were trained. The first three are hard-coded, the rest are pre-trained PPO policies. All DRL policies receive the agent’s 3D velocity vector as input (which is standard in Animal-AI) in addition to those described below. All DRL policies received a small negative reward of 0.05 for going backwards to disincentivise sub-optimal policies. No hyperparameter tuning was necessary to achieve sufficient performance. We used the ml-agents implementation of PPO (Juliani et al., 2018). All options were trained with the following hyperparameters.

**AnimalAI:**

```plaintext
trainer: ppo
epsilon: 0.2
lambda: 0.95
learning_rate: 1e-4
learning_rate_schedule: linear
memory_size: 128
normalize: false
sequence_length: 64
summary_freq: 10000
use_recurrent: false
vis_encode_type: simple
time_horizon: 128
batch_size: 64
buffer_size: 2024
hidden_units: 256
num_layers: 1
beta: 1.0e-2
max_steps: 1.0e7
num_epoch: 3
reward_signals:
  extrinsic:
    strength: 1.0
    gamma: 0.99
```

**B.1 Observe**

**Inputs** None.

**Description** The agent does not move, but keeps track of which direction objects are moving in the environment. If a blackout happens during the observation, the option will wait until the blackout is finished to return.

**Training** None.
B.2 Rotate

**Inputs** None.

**Description** Rotate clockwise until an object is seen.

**Training** None.

---

B.3 Drop

**Inputs** A boolean indicating left or right.

**Description** Move in a diagonal line left or right until the agent falls off the platform (as monitored via the vertical velocity component).

**Training** None.

---

B.4 Interact

**Inputs** The bounding box of a single object.

**Description** Go touch an object.

**Training** The agent and a goal of varying sizes are placed at random in the arena.

---

B.5 Explore

**Inputs** The bounding box of a single object.

**Description** Go around an object clockwise or anti-clockwise.

**Training** The agent is always placed facing a wall of various dimensions and multiple goals are behind the wall. However, the agent does not ‘see’ the goals at all so it will learn the policy of wall-following until it bumps into the goal.

---

B.6 Collect

**Inputs** The masked image for orange goals.

**Description** Collect as many orange goals as possible in an efficient manner.

**Training** The agent is placed in an arena with a random number of randomly sized orange goals.

---

B.7 Avoid

**Inputs** The full RGB image.

**Description** Avoid red objects while going to touch goal.

**Training** The training set is composed of many variants of the agent needing to reach a green goal while avoiding red lava or red balls which give negative reward and terminate the episode. The lava is placed in many different configurations to resemble mazes that the agent has to navigate. The goal is always visible to the agent.
B.8 Balance

**Inputs** The masked image for blue platforms and a single bounding box for the goal.

**Description** Balance on platform without falling off to get to the goal.

**Training** The training set is composed of various variations of L or U shaped platforms where the agent is on one side and the goal on the other. The floor is all lava to terminate episodes as soon as the agent falls.

B.9 Climb

**Inputs** The masked image for pink ramps.

**Description** Climb up ramp.

**Training** The agent is placed in an arena with a single ramp of varying dimensions with a goal at the top. The agent does not ‘see’ the goal as with the explore training. The agent is given a positive reward proportional to its upwards velocity to accelerate training.

C Detect Heuristics

To translate a list of bounding boxes into a logic program composed of objects and their relations we employ a few simple heuristics described below.

**On**(x,y). X can be the agent or goal. Y can be any object in AAI. To determine whether a goal is on top of an object y we take the bounding box the size of the goal, mirror it downwards and check if its colour is other than that of the floor. To check if the agent is on an object we take the bottom quartile of the agent’s view and check if there is at least a 70% overlap with another bounding box in the scene.

**Danger**. If the column beneath a goal’s bounding box intersects any red objects then there is said to be a danger in getting to the goal.

**Adjacent**. Two objects are adjacent if the shortest distance between their bounding boxes is 0.

**More_goals**(side). To determine whether more goals are on the left or the right we split the screen in two and count how many goal centroids are on each side. If they end up being equal we select the side with the larger occlusion area.

D IMP Training Set

In this section we present the training set used to learn a general meta-policy to solve the AAI testbed. The choice of arena for each category is defined by two criteria:

- The arena teaches the high-level lesson necessary for a whole category. For example, prefer bigger goals for y-mazes or choose sides with more goals for numerosity.
Table 1: DUA’s performance on AAI categories of interest. Average ± s.d. over 10 runs.

<table>
<thead>
<tr>
<th>Category</th>
<th>Best</th>
<th>Top 10 Average</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Food and Obstacles</td>
<td>89</td>
<td>53</td>
<td>99±4</td>
</tr>
<tr>
<td>Moving Food</td>
<td>77</td>
<td>71</td>
<td>70±5</td>
</tr>
<tr>
<td>Unreachable Food</td>
<td>100</td>
<td>72</td>
<td>81±12</td>
</tr>
<tr>
<td>Multiple Food Stationary</td>
<td>77</td>
<td>47</td>
<td>20±11</td>
</tr>
<tr>
<td>Multiple Food Moving</td>
<td>55</td>
<td>28</td>
<td>23±13</td>
</tr>
<tr>
<td>Avoid Red</td>
<td>50</td>
<td>19</td>
<td>57±5</td>
</tr>
<tr>
<td>Ramp Usage</td>
<td>37</td>
<td>9</td>
<td>45±4</td>
</tr>
<tr>
<td>Hot zones</td>
<td>75</td>
<td>62</td>
<td>81±5</td>
</tr>
<tr>
<td>Generalisation and Adaptability</td>
<td>54</td>
<td>36</td>
<td>13±2</td>
</tr>
<tr>
<td>Internal Models</td>
<td>57</td>
<td>44</td>
<td>22±2</td>
</tr>
<tr>
<td>Y-Mazes</td>
<td>88</td>
<td>76</td>
<td>78±3</td>
</tr>
<tr>
<td>Delayed Gratification</td>
<td>53</td>
<td>34</td>
<td>46±5</td>
</tr>
<tr>
<td>Detour Tasks</td>
<td>33</td>
<td>12</td>
<td>10±2</td>
</tr>
<tr>
<td>Cylinder Tasks</td>
<td>77</td>
<td>48</td>
<td>19±5</td>
</tr>
<tr>
<td>Thorndike Escape Experiments</td>
<td>50</td>
<td>27</td>
<td>21±6</td>
</tr>
<tr>
<td>T-Maze</td>
<td>100</td>
<td>66</td>
<td>48±12</td>
</tr>
<tr>
<td>Spatial Elimination</td>
<td>26</td>
<td>17</td>
<td>48±5</td>
</tr>
<tr>
<td>Support and Gravity Bias</td>
<td>44</td>
<td>31</td>
<td>30±6</td>
</tr>
<tr>
<td>Radial Mazes</td>
<td>59</td>
<td>29</td>
<td>19±4</td>
</tr>
<tr>
<td>Object Permanence</td>
<td>25</td>
<td>9</td>
<td>29±3</td>
</tr>
<tr>
<td>Numerosity</td>
<td>50</td>
<td>43</td>
<td>64±3</td>
</tr>
<tr>
<td>Tool use</td>
<td>11</td>
<td>2</td>
<td>0±0</td>
</tr>
<tr>
<td>Overall</td>
<td>43.7</td>
<td>32.9</td>
<td>39.0±0.6</td>
</tr>
</tbody>
</table>

The agent succeeds the episode every time it chooses the right sequence of options. In other words, ensure that the arenas are easy enough that the DRL options reliably succeed when chosen in the appropriately. Otherwise this will lead to unnecessary noise in the learning task.

**Basic** In this arena the agent a goal and a wall are randomly placed in the arena. The agent must learn how to explore effectively and navigate to goals when visible. Macro-steps are limited to 3. Category: *Basic Food and Obstacles.*
Lava In this arena lava is placed between the agent and the visible goal. The agent must learn to avoid lava to reach the goal. Macro-steps are limited to 1. Category: Avoid Red.

Ramp This arena contains a ramp, an L-shaped platform and a goal above the platform. The agent must first climb the ramp, then balance on the platform to reach the goal. Macro-steps are limited to 2. Category: Ramp Usage.

Preference This arena contains a ramp, an L-shaped platform and a goal above the platform. The agent must first climb the ramp, then balance on the platform to reach the goal. Macro-steps are limited to 2. Category: Y-Mazes.

Count There are two goals on the left and one on the right. The agent must first learn to drop on the left side and collect the multi goals. Macro-steps are limited to 2. Category: Numerosity.

Spatial Elimination The agent has a forced choice between two occluding objects likely to hide the goal. The agent must learn to preferentially explore objects most likely to occlude the goal. Macro-steps are limited to 2. Category: Spatial Elimination.

Object Permanence The episode starts with the goal moving towards the back of the arena and then there is a blackout following which the goal is hidden behind the occluding object. The agent must learn to wait for the blackout and understand that the goal has not vanished, but must be behind the one occluding object. Macro-steps are limited to 3. Category: Object Permanence.