1
Apperception

1.1 Introduction

'Apperception' (from the French 'apercevoir') is a term introduced by Leibniz in the *New Essays Concerning Human Understanding* (Leibniz, 1765), and is a central term in Kant's *Critique of Pure Reason* (Kant, 1781). Nowadays, the term has two distinct but related meanings: first, to apperceive something is to be self-consciously aware of it; second, to apperceive something is to assimilate it into a larger collection of ideas: to make sense of something by unifying it with one's other thoughts into a coherent whole. In this paper, I focus on the second sense.

Apperception should be distinguished from mere low-level perception. Consider a neural network trained to classify images into classes. Although the network can perceive the image as a dog, *it has no idea what a dog actually is*. The network may be able to classify the image under the label 'dog', but it does not understand what 'dog' means, since it does not understand the *inferential connections* between the concept 'dog' and other concepts: it does not know that dogs are mammals, that no dog is also a cat, or that dogs are (typically) loyal to their masters. In other words, the neural network may be able to perceive, but it is not able to apperceive, since it does not understand the inferential relations between concepts.

To apperceive, then, is to make sense of one's sensory inputs by combining them into a unified whole using concepts and inferential relations between concepts.

In this paper, I describe a computer model of apperception. This builds on previous work (Evans, Hernandez-Orallo, Welbl, Kohli and Sergot, 2019). Our system, the Apperception Engine takes as input a sensory sequence and constructs an explicit causal theory that both explains the sequence and also satisfies a set of unity conditions designed to ensure that the constituents of the theory – the objects, properties, and propositions – are combined together in a relational structure. In this previous work, we showed, in a range of experiments, how this system is able to outperform recurrent networks and other baselines on a range of tasks, including Hofstadter's *Seek Whence* dataset.

But in our initial implementation, there was one fundamental limitation: we assumed the sensory given had already been discretised. We assumed *some other system had already parsed the raw sensory input using a set of discrete categories*, so that all the Apperception Engine had to do was receive this already-digested discretised input, and make sense of it (Smith, 2019). But what if we don't have access to pre-parsed input? What if our sensory sequence is raw unprocessed information, say, a sequence of noisy pixel arrays from a video camera, for example?

The central contribution of this paper is a major extension of the Apperception Engine so that it can be applied to raw unprocessed sensory input. This involves two phases. First, we extend our system to receive ambiguous (but still discretised) input: sequences of disjunctions. Second, we use a neural network to map raw sensory input to disjunctive input. Our binary neural network is encoded as a logic program, so the weights of the network and the rules of the theory can be found *jointly* by solving a single SAT problem. This way, we are able to jointly learn how to perceive (mapping raw sensory information to concepts) and apperceive (combining concepts into declarative rules).

We test the Apperception Engine in the *Sokoban* domain, and show how the system is able to learn the game's dynamics from a sequence of noisy pixel arrays. This system is, to the best of our knowledge, the first system that is able to learn explicit provably correct dynamics of non-trivial games from raw pixel input.

1.2 Method

In describing the Apperception Engine, we shall use three increasingly complex forms of sequential input. We start by assuming that the sensory sequence has already been discretised into ground atoms of first-order logic representing sensor readings. So, for example, the (discrete) fact that light sensor a is *red* might be represented by the atom $red(a)$. Next, we expand to consider disjunctive input sequences. For example, to represent that sensor a is either *red* or *orange* we would represent our uncertainty by $red(a) \vee orange(a)$. Finally, we let go entirely of our simplifying assumption of already-discretised sensory input and consider sequences of *raw unprocessed input*. Consider, for example, a sequence of pixel arrays from a video camera.

1.2.1 Making sense of unambiguous symbolic input

Definition 1.1 *An* **unambiguous symbolic sensory sequence** *is a sequence of sets of ground atoms. Given a sequence $S = (S_1, S_2, ...)$, a* **state** *S_t is a set of ground atoms, representing a* partial *description of the world at a discrete time-step t.*

Example 1.1 Consider, for example, the following sequence $S_{1:10}$. Here there are two sensors a and b, and each sensor can be *on* or *off*.

$$
\begin{array}{ll}
S_1 = \{\} & S_2 = \{off(a), on(b)\} \\
S_3 = \{on(a), off(b)\} & S_4 = \{on(a), on(b)\} \\
S_5 = \{on(b)\} & S_6 = \{on(a), off(b)\} \\
S_7 = \{on(a), on(b)\} & S_8 = \{off(a), on(b)\} \\
S_9 = \{on(a)\} & S_{10} = \{\}
\end{array}
$$

Note that there is no expectation that a sensory sequence contains readings for all sensors at all time-steps. Some of the readings may be missing. In state S_5, we are missing a reading for a, while in state S_9, we are missing a reading for b. In states S_1 and S_{10}, we are missing sensor readings for both a and b.

The Apperception Engine makes sense of a sensory sequence by constructing a unified theory that explains that sequence. The key notions, here, are "theory", "explains", and "unified". We consider each in turn.

Definition 1.2 *A* **theory** *is a four-tuple* (ϕ, I, R, C) *where:*

- ϕ *is a type signature specifying the types of objects, variables, and arguments of predicates*
- I *is a set of initial conditions (ground atoms that are well-typed according to* ϕ*)*
- R *is a set of rules in Datalog*$^{\Rightarrow}$*, an extension of Datalog for representing causal rules*
- C *is a set of constraints*

There are two types of rule in Datalog$^{\Rightarrow}$*. A* **static rule** *is a definite clause of the form* $\alpha_1 \wedge ... \wedge \alpha_n \rightarrow \alpha_0$*, where* $n \geq 0$ *and each* α_i *is an* unground *atom consisting of a predicate and a list of variables. This means: if conditions* $\alpha_1, ...\alpha_n$ *hold at the current time-step, then* α_0 *also holds at that time-step. A* **causal rule** *is a clause of the form* $\alpha_1 \wedge ... \wedge \alpha_n \Rightarrow \alpha_0$*, where* $n \geq 0$ *and each* α_i *is an unground. A causal rule expresses how facts change over time.* $\alpha_1 \wedge ... \wedge \alpha_n \Rightarrow \alpha_0$ *states that if conditions* $\alpha_1, ...\alpha_n$ *hold at the current time-step, then* α_0 *will be true at the* next *time-step. So, for example,* $on(X) \Rightarrow \mathit{off}(X)$ *states that if object* X *is currently on, then* X *will become off at the next-time-step.*

There are three types of constraint. A **unary xor constraint** *is an expression of the form* $\forall X{:}t, p_1(X) \oplus ... \oplus p_n(X)$ *where* $n > 1$*. Here,* $\oplus$ *means exclusive-or, so for example* $\forall X{:}t, p_1(X) \oplus p_2(X)$ *means that for every object* X *of type* t*,* X *either satisfies* p_1 *or* p_2*, but not both. A* **binary xor constraint** *is an expression of the form* $\forall X{:}t_1, \forall Y{:}t_2, r_1(X,Y) \oplus ... \oplus r_n(X,Y)$ *where* $n > 1$*. A* **uniqueness constraint** *is an expression of the form* $\forall X{:}t_1, \exists!Y{:}t_2, r(X,Y)$*, which means that for all objects* X *of type* t_1 *there exists a* unique *object* Y *of type* t_2 *such that* $r(X,Y)$*.*

Example 1.2 Consider the frame $\phi = (T, O, P, V)$, consisting of types $T = \{s\}$, objects $O = \{a{:}s, b{:}s\}$, predicates $P = \{on(s), \mathit{off}(s), r(s,s), p_1(s), p_2(s), p_3(s)\}$, and variables $V = \{X{:}s, Y{:}s\}$. Consider the theory $\theta = (\phi, I, R, C')$, where:

$$I = \left\{ \begin{array}{l} p_1(b) \\ p_2(a) \\ r(a,b) \\ r(b,a) \end{array} \right\} \quad R = \left\{ \begin{array}{l} p_1(X) \Rightarrow p_2(X) \\ p_2(X) \Rightarrow p_3(X) \\ p_3(X) \Rightarrow p_1(X) \\ p_1(X) \rightarrow on(X) \\ p_2(X) \rightarrow on(X) \\ p_3(X) \rightarrow \mathit{off}(X) \end{array} \right\} \quad C' = \left\{ \begin{array}{l} \forall X{:}s,\ on(X) \oplus \mathit{off}(X) \\ \forall X{:}s,\ p_1(X) \oplus p_2(X) \oplus p_3(X) \\ \forall X{:}s,\ \exists!Y{:}s\ r(X,Y) \end{array} \right\}$$

Definition 1.3 *Every theory* $\theta = (\phi, I, R, C)$ *generates an infinite sequence* $\tau(\theta)$ *of sets of ground atoms, called the* **trace** *of that theory. Here,* $\tau(\theta) = (A_1, A_2, ...)$*, where each* A_t *is the smallest set of atoms satisfying the following conditions:*

- $I \subseteq A_1$
- *If there is a static rule* $\beta_1 \wedge ... \wedge \beta_m \rightarrow \alpha$ *in* R *and a ground substitution* σ *such that* A_t *satisfies* $\beta_i[\sigma]$ *for each antecedent* β_i*, then* $\alpha[\sigma] \in A_t$
- *If there is a causal rule* $\beta_1 \wedge ... \wedge \beta_m \Rightarrow \alpha$ *in* R *and a ground substitution* σ *such that* A_t *satisfies* $\beta_i[\sigma]$ *for each antecedent* β_i*, then* $\alpha[\sigma] \in A_{t+1}$

- Frame axiom*: if α is in A_{t-1} and there is no atom in A_t that is incompossible with α w.r.t constraints C, then $\alpha \in A_t$. Two ground atoms are* **incompossible** *if they are ruled out by one of the constraints in C.*

Note that the state transition function is deterministic: A_{t+1} is uniquely determined by A_t.

Example 1.3 The infinite trace $\tau(\theta) = (A_1, A_2, ...)$ for the theory θ of Example 1.2 begins with:

$$\begin{aligned}
A_1 &= \{on(a), on(b), p_2(a), p_1(b), r(a,b), r(b,a)\} \\
A_2 &= \{off(a), on(b), p_3(a), p_2(b), r(a,b), r(b,a)\} \\
A_3 &= \{on(a), off(b), p_1(a), p_3(b), r(a,b), r(b,a)\} \\
A_4 &= \{on(a), on(b), p_2(a), p_1(b), r(a,b), r(b,a)\} \\
&...
\end{aligned}$$

Note that the trace repeats at step 4. It is possible to show in general that the trace always repeats after some finite set of time steps, since the set of ground atoms is finite and the state transition of Definition 1.3 is Markov.

Now that we have defined what we mean by a "theory", next we define what it means for a theory to "explain" a sensory sequence.

Definition 1.4 *Given two finite sequences $S = (S_1, ..., S_T)$ and $S' = (S'_1, ..., S'_{T'})$, define $\sqsubseteq$ as:*

$$S \sqsubseteq S' \text{ iff } T \leq T' \text{ and } \forall 1 \leq i \leq t,\ S_i \subseteq S'_i$$

So far, we assume both sequences are finite. Now we extend this partial order so that S' may be infinitely long: $S \sqsubseteq S'$ if S' has a finite subsequence S'' such that $S \sqsubseteq S''$. If $S \sqsubseteq S'$, we say that S is **covered by** *S', or that S' covers S (Lee and De Raedt, 2004; Tamaddoni-Nezhad and Muggleton, 2009).*

A theory θ **explains** *a sensory sequence S if the trace of θ covers S, i.e. $S \sqsubseteq \tau(\theta)$.*

Example 1.4 The theory θ of Example 1.2 explains the sensory sequence S of Example 1.1, since the trace $\tau(\theta)$ (as shown in Example 1.3) covers S. Note that $\tau(\theta)$ "fills in the blanks" in the original sequence S, both predicting final time-step 10 and retrodicting initial time-step 1.

Next, we proceed from explaining a sensory sequence to "making sense" of that sequence. In providing a theory θ that explains a sensory sequence S, we make S intelligible by placing it within a bigger picture: while S is a scanty and incomplete description of a fragment of the time-series, $\tau(\theta)$ is a complete and determinate description of the whole time-series.

In order for θ to make sense of S, it is *necessary* that $\tau(\theta)$ covers S. But this condition is not, on its own, *sufficient*. The extra condition that is needed for θ to count as "making sense" of S is for θ to be *unified*. In this paper, we formalize what

it means for a theory to be "unified" using Kant's key notion of the "synthetic unity of apperception"[1] [2].

Definition 1.5 *A trace $\tau(\theta)$ is (i) a sequence of (ii) sets of ground atoms composed of (iii) predicates and (iv) objects*[3]*. For the theory θ to be* **unified** *is for unity to be achieved at each of these four levels:*

- *Objects are united in space*[4]*. A theory θ satisfies* **spatial unity** *if for each state A_i in $\tau(\theta) = (A_1, A_2, ...)$, for each pair (x, y) of distinct objects, x and y are connected via a chain of binary atoms $\{r_1(x, z_1), r_2(z_1, z_2), ...r_n(z_{n-1}, z_n), r_{n+1}(z_n, y)\} \subseteq A_i$*
- *Predicates are united via constraints*[5]*. A theory $\theta = (\phi, I, R, C)$ satisfies* **conceptual unity** *if for each unary predicate p in ϕ, there is some xor constraint in C of the form $\forall X{:}t, p(X) \oplus q(X) \oplus ...$ containing p; and, for each binary predicate r in ϕ, there is some xor constraint in C of the form $\forall X{:}t_1, \forall Y{:}t_2, r(X, Y) \oplus s(X, Y) \oplus ...$ or some $\exists!$ constraint in C of the form $\forall X{:}t, \exists! Y{:}t_2, r(X, Y)$.*
- *Ground atoms are united into sets (states) by jointly respecting constraints and static rules*[6]*. A theory $\theta = (\phi, I, R, C)$ satisfies* **static unity** *if every state $(A_1, A_2, ...)$ in $\tau(\theta)$ satisfies all the constraints in C and is closed under the static rules in R.*
- *States (sets of ground atoms) are united into a sequence by causal rules*[7]*. A sequence $(A_1, A_2, ...)$ of states satisfies* **temporal unity** *with respect to a set $R_{\Rightarrow}$ of causal rules if, for each $\alpha_1 \wedge ... \wedge \alpha_n \Rightarrow \alpha_0$ in $R_{\Rightarrow}$, for each ground substitution σ, for each time-step t, if $\{\alpha_1\sigma, ..., \alpha_n\sigma\} \subseteq A_t$ then $\alpha_0\sigma \in A_{t+1}$. Note that, from the definition of the trace in Definition 1.3, the trace $\tau(\theta)$* automatically *satisfies temporal unity.*

Example 1.5 The theory θ of Example 1.2 satisfies the four unity conditions since:

- For each state A_i in $\tau(\theta)$, a is connected to b via the singleton chain $\{r(a, b)\}$, and b is connected to a via $\{r(b, a)\}$.
- The predicates of θ are $on, off, p_1, p_2, p_3, r$. Here, *on* and *off* are involved in the constraint $\forall X{:}s,\ on(X) \oplus off(X)$, while p_1, p_2, p_3 are involved in the constraint $\forall X{:}s,\ p_1(X) \oplus p_2(X) \oplus p_3(X)$, and r is involved in the constraint $\forall X{:}s,\ \exists! Y{:}s\ r(X, Y)$.
- Let $\tau(\theta) = (A_1, A_2, A_3, A_4, ...)$. It is straightforward to check that A_1, A_2, and A_3 satisfy each constraint in C. Observe that A_4 repeats A_1, and the dynamics are

[1] In this paper, we do not focus on Kant exegesis, but do provide some key references. All references to the *Critique of Pure Reason* use the standard [A/B] reference system, where A refers to the First Edition (1781), and B refers to the Second Edition (1787).

[2] "The principle of the synthetic unity of apperception is the supreme principle of all use of the understanding" [B136]; it is "the highest point to which one must affix all use of the understanding, even the whole of logic and, after it, transcendental philosophy" [B134].

[3] Strictly speaking, we mean two *constants* denoting two objects. Throughout, we follow the common practice of treating the Herbrand interpretation as a distinguished interpretation, blurring the difference between a constant and the object denoted by the constant.

[4] See [B203]. See also the *Third Analogy* [A211-215/B256-262].

[5] See [A103-11]. See also: "What the form of disjunctive judgment may do is contribute to the acts of forming categorical and hypothetical judgments the perspective of their possible *systematic unity*", (Longuenesse, 1998, p.105)

[6] See the *schema of community* [A144/B183-4].

[7] See the *schema of causality* [A144/B183].

Markov, so all subsequent states are copies of A_1, A_2, and A_3. Hence, every state in the infinite sequence $(A_1, A_2, A_3, ...)$ satisfies each constraint.
- Temporal unity is automatically satisfied by the definition of the trace $\tau(\theta)$ in Definition 1.3

Now we are ready to define the key notion of this paper.

Definition 1.6 *A theory θ* **makes sense** *of a sensory sequence S if θ explains S, i.e. $S \sqsubseteq \tau(\theta)$, and θ satisfies the four conditions of unity of Definition 1.5.*

Example 1.6 The theory θ of Example 1.2 explains the sensory sequence S of Example 1.1, since $S \sqsubseteq \tau(\theta)$ (Example 1.4) and θ satisfies the four conditions of unity (Example 1.5).

Definition 1.7 *The input to an apperception task is a triple (S, ϕ, C) consisting of a sensory sequence S, a suitable frame ϕ, and a set C of (well-typed) constraints such that (i) each predicate in S appears in some constraint in C and (ii) S can be extended to satisfy C: there exists a sequence S' covering S such that S'_i satisfies each constraint in C.*

Given such an input triple (S, ϕ, C), the **simple apperception task** *is to find a lowest cost theory $\theta = (\phi', I, R, C')$ such that ϕ' extends ϕ, $C' \supseteq C$, and θ makes sense of S. Here, cost(θ) is just the total number of ground atoms in I plus the total number of unground atoms in the rules of R.*

Example 1.7 Recall that theory θ of Example 1.2 makes sense of the sequence from Example 1.1. Now consider an alternative theory based on the type signature $\phi_2 = (T_2, O_2, P_2, V_2)$, where $T_2 = \{s\}$, $O_2 = \{a{:}s, b{:}s, c{:}s\}$, $P_2 = \{on(s), off(s), right(s, s)\}$, and $V_2 = \{X{:}s, Y{:}s\}$. The alternative theory $\theta_2 = (\phi_2, I_2, R_2, C_2)$ where

$$I_2 = \left\{ \begin{array}{lll} on(a) & on(b) & off(c) \\ right(a,b) & right(b,c) & right(c,a) \end{array} \right\}$$

$$R_2 = \left\{ \begin{array}{l} right(X,Y) \wedge off(X) \supset off(Y) \\ right(X,Y) \wedge on(X) \supset on(Y) \end{array} \right\}$$

$$C_2 = \left\{ \begin{array}{l} \forall X{:}sensor,\ on(X) \oplus off(X) \\ \forall X{:}sensor,\ \exists! Y{:}sensor,\ right(X,Y) \end{array} \right\}$$

Now θ_2 also makes sense of the sensory sequence S from Example 1.1. But while θ used three invented predicates (p_1, p_2, and p_3), θ_2 makes use of an invented *object*, c, postulated to explain the sensory sequence more concisely. Note that while θ has cost 16, θ_2 has cost 12.

1.2.2 The Apperception Engine

Next, we describe how our system, the Apperception Engine, solves apperception tasks. Given as input an apperception task (S, ϕ, C), the engine searches for a frame ϕ' and a theory $\theta = (\phi', I, R, C')$ where ϕ' extends ϕ, $C' \supseteq C$ and θ is a unified interpretation of S.

Definition 1.8 *A* **template** *is a structure for circumscribing a large but finite set of theories. It is a frame together with constants that bound the complexity of the rules in the theory. Formally, a template χ is a tuple $(\phi, N_{\rightarrow}, N_{\supset}, N_B)$ where ϕ is a frame, $N_{\rightarrow}$ is the max number of static rules allowed in R, $N_{\supset}$ is the max number of causal rules allowed in R, and N_B is the max number of atoms allowed in the body of a rule in R.*

Our method, presented in Algorithm 1, is an anytime algorithm that enumerates templates of increasing complexity. For each template χ, it finds the theory θ with lowest cost that satisfies the conditions of unity. If it finds such a θ, it stores it. When it has run out of processing time, it returns the lowest cost θ it has found from all the templates it has considered.

Note that the relationship between the complexity of a template and the cost of a theory satisfying the template is not always simple. Sometimes a theory of lower cost may be found from a template of higher complexity. This is why we cannot terminate as soon as we have found the first theory θ. We must keep going, in case we later find a lower cost theory from a more complex template.

Algorithm 1: The APPERCEPTION ENGINE algorithm in outline

input : (S, ϕ, C), an apperception task
output: θ^*, a unified interpretation of S

1 $(s^*, \theta^*) \leftarrow (max(float), nil)$
2 **foreach** *template* χ *extending* ϕ *of increasing complexity* **do**
3 $\quad \theta \leftarrow \text{argmin}_\theta\{cost(\theta) \mid \theta \in \Theta_{\chi,C}, S \sqsubseteq \tau(\theta), unity(\theta)\}$
4 $\quad$ **if** $\theta \neq nil$ **then**
5 $\quad\quad s \leftarrow cost(\theta)$
6 $\quad\quad$ **if** $s < s^*$ **then**
7 $\quad\quad\quad (s^*, \theta^*) \leftarrow (s, \theta)$
8 $\quad\quad$ **end**
9 $\quad$ **end**
10 $\quad$ **if** *exceeded processing time* **then**
11 $\quad\quad$ **return** θ^*
12 $\quad$ **end**
13 **end**

The "heavy lifting" occurs in line 3, where we find, for a given template χ, a lowest cost theory θ satisfying χ that both explains the given sensory sequence S and also satisfies the four unity conditions. In order to jointly abduce a set I (of initial conditions) and induce sets R and C (of rules and constraints), we implement a Datalog$^{\supset}$ interpreter in Answer Set Programming (ASP). This interpreter takes a set I of atoms (represented as a set of ground ASP terms) and sets R and C of rules and constraints (represented again as a set of ground ASP terms), and computes the trace of the theory $\tau(\theta) = (S_1, S_2, ...)$ up to a finite time limit.

Concretely, we implement the Datalog$^{\ni}$ interpreter as an ASP program that computes $\tau(\theta)$ for theory θ. We implement the conditions of unity as ASP constraints, and we implement the cost minimization as an ASP program that counts the number of atoms in each rule and in each initialisation atom in I, and uses an ASP weak constraint (Calimeri, Faber, Gebser, Ianni, Kaminski, Krennwallner, Leone, Ricca and Schaub, 2012) to minimize this total. Then we generate ASP programs representing the sequence S, the initial conditions, the rules and constraints. We combine the ASP programs together and ask the ASP solver (`clingo` (Gebser, Kaminski, Kaufmann and Schaub, 2014)) to find a lowest cost solution. There may be multiple solutions that have equally lowest cost; the ASP solver chooses one of the optimal answer sets. We extract a readable interpretation θ from the ground atoms of the answer set. We do not go into more detail, for reasons of space. The source code for the APPERCEPTION ENGINE is available at `https://github.com/RichardEvans/apperception`.

1.2.3 Making sense of disjunctive symbolic input

In Section 1.2.1, the sensory sequence $(S_1, ..., S_T)$ was a sequence of sets of unambiguous discrete sensor readings. Each state S_i may be partial and incomplete, but all information that is present is unambiguous. In this section, we extend the APPERCEPTION ENGINE to handle disjunctive sensory input.

Definition 1.9 *A* **disjunctive input sequence** *is a sequence of sets of disjunctions of ground atoms.*

A disjunctive input sequence generalises the input sequence of Section 1.2.1 to handle uncertainty. Now if we are unsure if a sensor a satisfies predicate p or predicate q, we can express our uncertainty as $p(a) \vee q(a)$.

Example 1.8 Consider, for example, the following sequence $D_{1:10}$. This is a disjunctive variant of the unambiguous sequence from Example 1.1. Here there are two sensors a and b, and each sensor can be *on* or *off*.

$$
\begin{array}{ll}
D_1 = \{\} & D_2 = \{\mathit{off}(a), \mathit{on}(b)\} \\
D_3 = \{\mathit{on}(a), \mathit{off}(b)\} & D_4 = \{\mathit{on}(a), \mathit{on}(b)\} \\
D_5 = \{\mathit{off}(a) \vee \mathit{on}(a), \mathit{on}(b)\} & D_6 = \{\mathit{on}(a), \mathit{off}(b)\} \\
D_7 = \{\mathit{on}(a), \mathit{on}(b)\} & D_8 = \{\mathit{off}(a), \mathit{on}(b)\} \\
D_9 = \{\mathit{off}(a) \vee \mathit{on}(a)\} & D_{10} = \{\}
\end{array}
$$

$D_{1:10}$ contains less information than $S_{1:10}$ from Example 1.1, since D_9 is unsure whether a is *on* or *off*, while in S_9 a is *on*.

Recall that the $\sqsubseteq$ relation describes when one sequence is explained by another. We extend the $\sqsubseteq$ relation to handle disjunctive input sequences in the first argument.

Definition 1.10 *Let $D = (D_1, ..., D_T)$ be a disjunctive input sequence and S be an input sequence. $D \sqsubseteq S$ if S contains a finite subsequence $(S_1, ..., S_T)$ such that $S_i \models D_i$ for all $i = 1..T$.*

Example 1.9 The theory θ of Example 1.2 explains the disjunctive sensory sequence D of Example 1.8, since the trace $\tau(\theta)$ (as shown in Example 1.3) covers D.

The disjunctive apperception task generalises the simple apperception task of Definition 1.7 to disjunctive input sequences.

Definition 1.11 *The input to a disjunctive apperception task is a triple (D, ϕ, C) consisting of a disjunctive input sequence D, a suitable frame ϕ, and a set C of (well-typed) constraints such that (i) for each disjunction featuring predicates $p_1, ..., p_n$ there exists a constraint in C featuring each of $p_1, ..., p_n$. (ii) D can be extended to satisfy C.*

Given such an input triple (D, ϕ, C), the **disjunctive apperception task** *is to find a lowest cost theory $\theta = (\phi', I, R, C')$ such that ϕ' extends ϕ, $C' \supseteq C$, $D \sqsubseteq \tau(\theta)$, and θ satisfies the four unity conditions of Definition 1.5.*

1.2.4 Making sense of raw input

The reason for introducing the disjunctive apperception task is as a stepping stone to the real task of interest: making sense of sequences of *raw uninterpreted sensory input*. Suppose, for example, our input is a sequence of pixel arrays from a video camera.

Definition 1.12 *Let $\mathcal{R}$ be the set of all possible raw inputs. A* **raw input sequence** *of length T is a sequence in $\mathcal{R}^T$.*

A raw apperception framework uses a neural network $\pi_{\mathbf{w}}$, parameterised by weights $\mathbf{w}$, to map raw subregions of each $\mathbf{r}_i$ into K classes. Then the results of the neural network are transformed into a disjunction of ground atoms. Thus we transform the raw input sequence into a disjunctive input sequence.

Definition 1.13 *A* **raw apperception framework** *is a tuple $(\pi_{\mathbf{w}}, K, \Delta, \phi, C)$, where:*

- *$\pi_{\mathbf{w}}$ is a perceptual classifier, a multilabel classifier mapping subregions $\mathbf{p}_i^j$ of $\mathbf{r}_i$ to subsets of $\{1, ..., K\}$; π is parameterised by weight vector $\mathbf{w}$*
- *K is the number of classes that the perceptual classifier $\pi_{\mathbf{w}}$ uses*
- *Δ is a "disjunctifier", taking the raw input sequence $(\mathbf{r}_1, ..., \mathbf{r}_T)$ and producing a sequence of sets of disjunctions $(D_1, ..., D_T)$; Δ works by repeatedly applying the perceptual classifier $\pi_{\mathbf{w}}$ to the N subregions $\{\mathbf{p}_i^1, ..., \mathbf{p}_i^N\}$ of $\mathbf{r}_i$, transforming each result (a subset of $\{1, ..., K\}$) into a disjunction of ground atoms*
- *ϕ is a type signature*
- *C is a set of constraints*

The input to a raw apperception task is a raw sequence $r = (r_1, ..., r_T)$ together with a raw apperception framework $(\pi_{\mathbf{w}}, K, \Delta, \phi, C)$. Given sequence $r = (r_1, ..., r_T)$ and framework $(\pi_{\mathbf{w}}, K, \Delta, \phi, C)$, the **raw apperception task** *is to find the lowest cost weights w and theory θ such that θ is a solution to the disjunctive apperception task $((D_1, ..., D_T), \phi, C)$ where $D_i = \Delta(\pi_{\mathbf{w}}(\mathbf{p}_i^1), ..., \pi_{\mathbf{w}}(\mathbf{p}_i^N))$.*

The best $(\theta, \mathbf{w})$ *pair is:*

$$\underset{\theta,\mathbf{w}}{\operatorname{argmax}} \log p(\theta) + \sum_{k=1}^{K} |\{\mathbf{p} \in \mathbf{P} \mid k \in \pi_{\mathbf{w}}(\mathbf{p})\}| \cdot \log \frac{1}{|\{\mathbf{p} \in \mathbf{P} \mid k \in \pi_{\mathbf{w}}(\mathbf{p})\}|}$$

where $\mathbf{P}$ *is the set of all subregions* $\mathbf{p}_i^j$ *in each* $\mathbf{r}_i$*.*

The intuition here is that $p(\theta) = 2^{-len(\theta)}$ represents the prior probability of the theory θ, while the other term penalises the neural network $\pi_{\mathbf{w}}$ for mapping many subregions to the same class. In other words, it prefers highly selective classes, minimising the number of subregions that are assigned by $\pi_{\mathbf{w}}$ to the same class. This particular optimisation can be justified on Bayesian grounds. We omit the justification for reasons of space.

1.2.5 Applying the Apperception Engine to raw input

In the experiments that follow, we use a binary neural network (Hubara, Courbariaux, Soudry, El-Yaniv and Bengio, 2016; Kim and Smaragdis, 2016; Rastegari, Ordonez, Redmon and Farhadi, 2016; Cheng, Nührenberg, Huang and Ruess, 2018; Narodytska, Kasiviswanathan, Ryzhyk, Sagiv and Walsh, 2018) for our parameterised perceptual classifier. Binary neural networks (BNNs) are increasingly popular because they are more efficient (both in memory and processing) than standard artificial neural networks. But our interest in BNNs is not so much in their resource efficiency as in their *discreteness*.

In the BNNs that we use (Cheng, Nührenberg, Huang and Ruess, 2018), the node activations and weights are all binary values in $\{0, 1\}$. If a node has n binary inputs $x_1, ..., x_n$, with associated binary weights $w_1, ..., w_n$, the node is activated if the total sum of inputs *xnor*-ed with their weights is greater than or equal to half the number of inputs. In other words, the node is activated if

$$\sum_{i=1}^{n} x_i \mathbin{\overline{\oplus}} w_i \geq \left\lceil \frac{n}{2} \right\rceil$$

Because the activations and weights are binary, the state of the network can be represented by a set of atoms, and the dynamics of the network can be defined as a logic program. This means we can combine the low-level perception task (of mapping raw data to concepts) and the high-level apperception task (of combining concepts into rules) *into a single logic program* in ASP, and solve both simultaneously using SAT.

1.3 Experiment: Sokoban

In this experiment, we combined the Apperception Engine with a neural network, simultaneously learning the weights of the neural network and also finding an interpretable theory that explains the sensory given. We used *Sokoban* as our domain. Here, the system is presented with a sequence of noisy pixel images together with associated actions. The system must jointly...

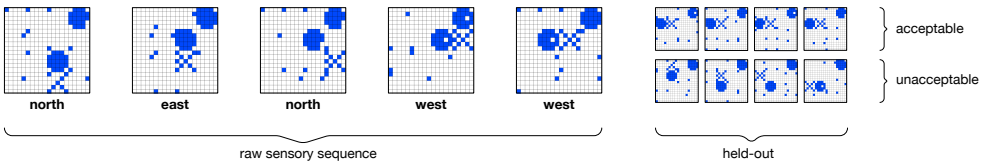

Fig. 1.1: The Sokoban task. The input is a sequence of (image, action) pairs. For the held-out time-step, there is a set of acceptable images, and a set of unacceptable images.

- parse the noisy pixel images into a set of persistent objects, with properties that change over time
- construct a set of rules that explain how the properties change over time as a result of the actions being performed

We wanted the learned dynamics to be 100% correct. Although next-step prediction models based on neural networks are able, with sufficient data, to achieve accuracy of 99%, this is insufficient for our purposes. If a learned dynamics model is going to be used for long-term planning, 99% is insufficiently accurate, as the roll-outs will become increasingly untrustworthy as we progress through time, since 0.99^t quickly approaches 0 as t increases (Buesing, Weber, Racaniere, Eslami, Rezende, Reichert, Viola, Besse, Gregor, Hassabis *et al.*, 2018).

1.3.1 The data

In this task, the raw input is a sequence of pairs containing a binarised 20×20 image together with a player action from the action space $\mathcal{A} = \{north, east, south, west\}$. In other words, $\mathcal{R} = \mathbb{B}^{20\times 20} \times \mathcal{A}$, and $(r_1, ..., r_T)$ is a sequence of (image, action) pairs from $\mathcal{R}$.

Each array is generated from a 4×4 grid of 5×5 sprites. Each sprite is rendered using a certain amount of noise (random pixel flipping), and so each 20×20 pixel image contains the accumulated noise from the various noisy sprite renderings.

Our trajectory contains a sequence of 17 (image, action) pairs, plus held-out data for evaluation. Because of the noisy sprite rendering process, there are many possible acceptable pixel arrays for time step 18. These acceptable pixel arrays were generated by taking the true underlying symbolic description of the Sokoban state at time step 18, and producing many renderings, using the noisy sprite rendering process described above. A set of unacceptable pixel arrays for time step 18 was generated by considering various symbolic states distinct from the true symbolic state at time step 18. For each such distinct symbolic state, a pixel rendering was generated. Figure 1.1 shows an example, but here the time-series is shorter to fit on the page.

In our evaluation, a model is considered accurate if it considers every acceptable pixel array at time 18 to be plausible, and considers every unacceptable pixel array at time 18 to be implausible. This is a stringent test. We do not give partial scores for getting some of the predictions correct.

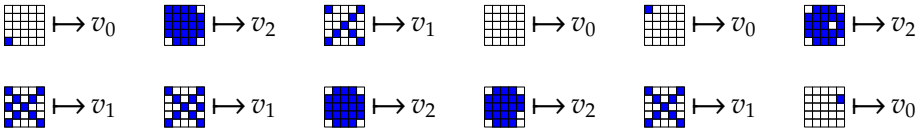

Fig. 1.2: A binary neural network maps sprite pixel arrays to types $\{v_0, v_1, v_2\}$.

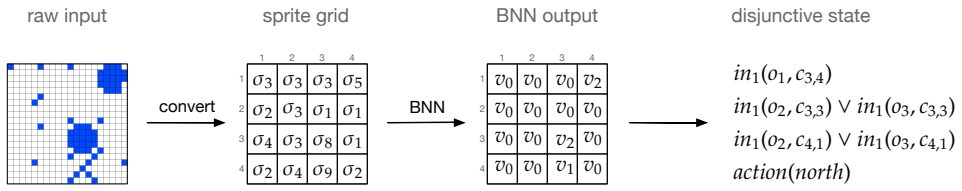

Fig. 1.3: A binary neural network converts the raw pixel input into a set of disjunctions. Here, there is one object o_1 of type v_1 and two objects o_2, o_3 of type v_2. If sprite σ is at (x, y), and $BNN(\sigma) = v_i$, then we create a disjunction featuring each object o of type v_i stating that any of them could be at (x, y).

1.3.2 The model

In outline, we convert the the raw input sequence into a disjunctive input sequence by imposing a grid on the pixel array and repeatedly applying a binary neural network to each sprite in the grid. In detail:

1. We impose a grid on the pixel array. We choose a sprite size k, and assume the pixel array can be divided into squares of size $k \times k$. We assume all objects fall exactly within grid cell boundaries. In this experiment, we set $k = 5$.
2. We choose a number m of persistent objects $o_1, ..., o_m$.
3. We choose a number n of distinct types of objects $v_1, ..., v_n$.
4. We add an additional type v_0, where v_0 is a distinguished identifier that will be used to indicate that there is *nothing* at a grid square.
5. We choose a total map $\kappa : \{o_1, ..., o_m\} \to \{v_1, ..., v_n\}$ from objects to types.
6. We apply a binary neural network (**BNN**) to each $k \times k$ sprite in the grid. The BNN implements a mapping $\mathbb{B}^{k \times k} \to \{v_0, v_1, ..., v_n\}$. If sprite σ is at (x, y), then $BNN(\sigma) = v_i$ can be interpreted as: it looks as if there is *some object* of type v_i at grid cell (x, y), for $i > 0$. If $BNN(\sigma) = v_0$, it means that there is *nothing* at (x, y). See Figure 1.2.
7. For each time-step, for each grid cell, we convert the output of the BNN into a disjunction of ground atoms: if sprite σ is at (x, y), and $BNN(\sigma) = v_i$, then we create a disjunction featuring each object o of type v_i stating that *any* of them could be at (x, y). See Figure 1.3.
8. We use the Apperception Engine to solve the disjunctive apperception task generated by steps 1 - 7.

In terms of the formalism of Section 1.2.4, the framework (π, ϕ, C) consists of:

- A perceptual classifier π that applies a binary neural network to each 5×5 sprite in the 4×4 grid, and uses the output of the binary network to produce a disjunction
- A frame $\phi = (T, O, P, V)$ consisting of $n+1$ types: $cell, v_1, ..., v_n$, and n predicates $in_i(v_i, cell)$, representing that an object of type v_i is (currently) in a particular cell
- A set of n constraints C insisting that every object of type v_i is always in exactly one cell

The input frame ϕ and initial constraints C are:

$$\phi = \begin{cases} T = \{cell, v_1, ..., v_n, d\} \\ O = \begin{cases} c(x,y) : cell \mid (x,y) \in 4 \times 4 \\ o_1{:}v_1, ..., o_m{:}v_n \\ north{:}d, east{:}d, south{:}d, west{:}d \end{cases} \\ P = \begin{cases} in_i(v_i, cell) \mid i = 1..n \\ action(d) \\ right(cell, cell) \\ below(cell, cell) \end{cases} \\ V = \{C{:}cell, A{:}d\} \cup \{X_i{:}v_i \mid i = 1..n\} \end{cases}$$

$$C = \begin{cases} \forall X_i{:}v_i,\ \exists!C{:}cell,\ in_i(X, C) \mid i = 1..n \\ \exists!A{:}d, action(A) \end{cases}$$

As background knowledge, we provide the spatial arrangement of the grid cells: $right(c_{1,1}, c_{2,1})$, $below(c_{1,1}, c_{1,2})$, etc.

The hyperparameters are n (the number of distinct types of object), m (the number of objects), and k (the sprite-size). We grid-search over n and m, and fix $k = 5$.

Both the Apperception Engine and the binary neural network are implemented as logic programs in ASP. Thus, we can find the binary weights of the neural network and the rules of our theory *simultaneously*, by solving a single SAT problem.

1.3.3 Understanding the interpretations

Figure 1.4 shows the best theory found by the Apperception Engine from one trajectory of 17 time-steps. On the left we show the raw input, the output of the neural network, and the set of atoms that are true at each time-step. On the right, we show the theory that explains the sequence. The top four rules of R describe how the man X moves when actions are performed. The middle four rules define four invented predicates $p_1, ...p_4$. These four unary predicates are used by the theory to describe when a block is being pushed in one of the four cardinal directions. The bottom four rules of R describe what happens when a block is being pushed in one of the four directions.

When neural network next-step predictors are applied to these sequences, the learned dynamics typically fail to generalise correctly to different-sized worlds or worlds

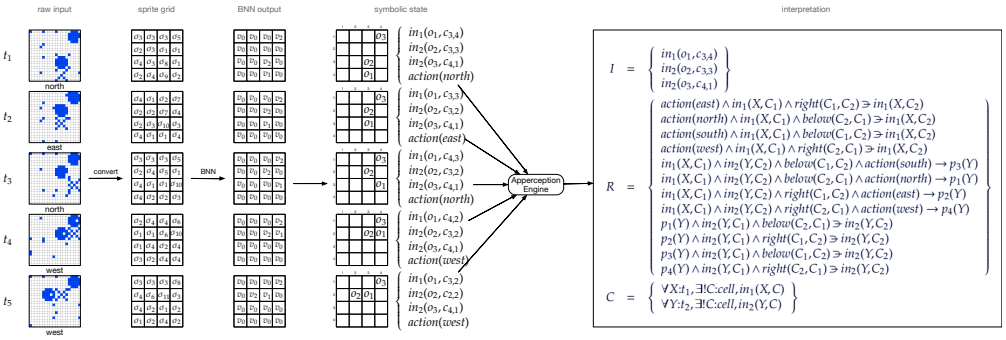

Fig. 1.4: Interpreting Sokoban from raw pixels. Raw input is converted into a sprite grid, which is converted into a grid of types v_0, v_1, v_2. The grid of types is converted into a disjunctive apperception task. The APPERCEPTION ENGINE finds a unified theory explaining the disjunctive input sequence, a theory which explains how objects' positions change over time.

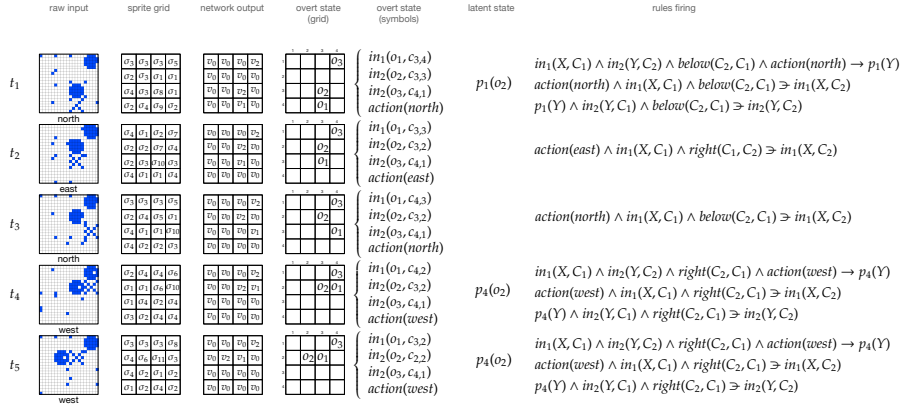

Fig. 1.5: The state evolving over time. Each row shows one time-step. We show the raw pixel input, the sprites, the output of the binary neural network, the set of ground atoms that are currently true, and the rules that fire.

with a different number of objects. But the theory leaned by the APPERCEPTION ENGINE applies to all Sokoban worlds, no matter how large, no matter how many objects. Not only is this learned theory correct, but it is provably correct.

Figure 1.5 shows the evolving state of the APPERCEPTION ENGINE over time. The grid on the left is the raw perceptual input, a grid of 20×20 pixels. The second element is a 4×4 grid of 5×5 sprites, formed by preprocessing the pixel array. The third element is the output of the binary neural network: a 4×4 grid of predicates v_0, v_1, v_2. If v_i is at (x, y), this means "it looks as if there is *some* object of type i at (x, y)" (but we don't yet know which particular object). So, for example, the grid in the top row states that there is some object of type 1 at $(3, 4)$, and some object of

type 2 at $(4, 1)$. Here, v_0 is a distinguished predicate meaning there is *nothing* at this grid square.

The fourth element is a 4×4 grid of persistent objects: if o_i is at (x, y) this means: the particular persistent object o_i is at (x, y). The fifth element is a set of ground atoms. This is a re-presentation of the persistent object grid (the fourth element) together with an atom representing the player's action. The sixth element shows the latent state. In Sokoban, the latent state stores information about which objects are being pushed in which directions. Here, in the top row, $p_1(o_2)$ means that persistent object o_2 is being pushed up. The seventh element shows which rules fire in which situations. In the top row, three rules fire. The first rule describes how the man moves when the *north* action is performed. The second rule concludes that a block is pushed northwards if a man is below the block and the man is moving north. The third rule describes how the block moves when it is pushed northwards.

Looking at how the engine interprets the sensory sequence, it is reasonable – in fact, we claim, inevitable – to attribute *beliefs* to the system. In the top row of Figure 1.5, for example, the engine believes the object at $(3, 3)$ is the same type of thing as the object at $(4, 1)$, but the object at $(3, 4)$ is not the same type of thing as the object at $(4, 1)$. As well as beliefs about a particular moment, the system also believes facts relating two successive moments. For example: the object at $(3, 4)$ at time t_1 is the very same persistent object as the thing that is at $(3, 3)$ at time t_2. As well as beliefs about particular situations, the system also has general beliefs that apply to all situations. For example: whenever the *north* action is performed, and the man is below a block, then the block is pushed upwards. One of the main reasons for using a purely declarative language (such as Datalog$^\ni$) as the target language of program synthesis is that an individual clause can be interpreted as a *judgement* with a truth-condition. If the program that generated the trace had been a procedural program, it would have been much less clear what judgement, if any, the procedure represented.

1.3.4 The baseline

Our hypothesis is that the strong inductive bias provided by the Datalog$^\ni$ language (Definition 1.2) and the unity constraints (Definition 1.5) should allow data-efficient learning of accurate models. To evaluate the hypothesis, we built a neural network baseline to compare data efficiency and accuracy.

The baseline we constructed for the Sokoban task is an auto-regressive model with a continuously-relaxed discrete bottleneck (Figure 1.6). The model applies an array of parameter-sharing multilayer perceptrons (MLPs) to each block of the game state, and concatenates the result with the one-hot representation of the actions before feeding it into an LSTM. The LSTM, combined with a dense layer, produces the parameters of Gumbel-Softmax continuous approximations of the categorical distribution, one per each block of the state. These distributions, when the model is learned well, can encode a close-to-symbolic representation of the current state without direct supervision. The step following is a decoder network consisting of a two-layer perceptron which targets the next raw state of the sequence.

Given that the presented model is a purely generative model over a large state space, in order to compare it to the Apperception Engine, we add a density es-

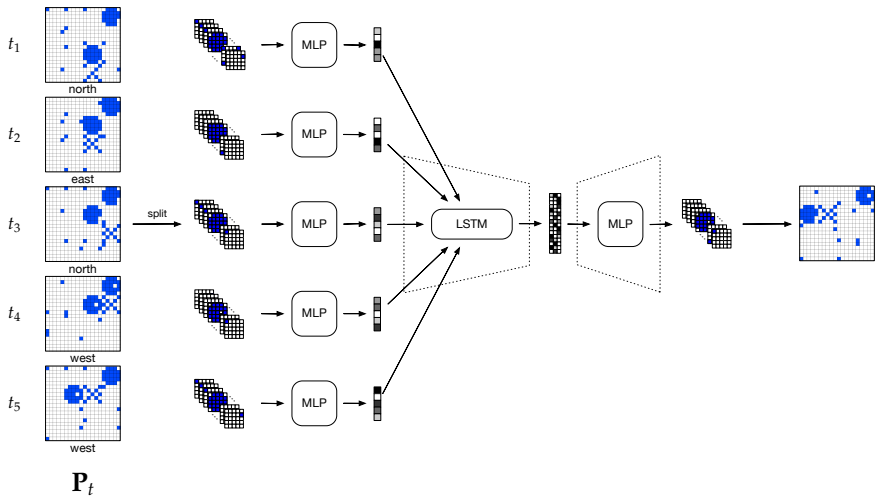

Fig. 1.6: The baseline model for the Sokoban task.

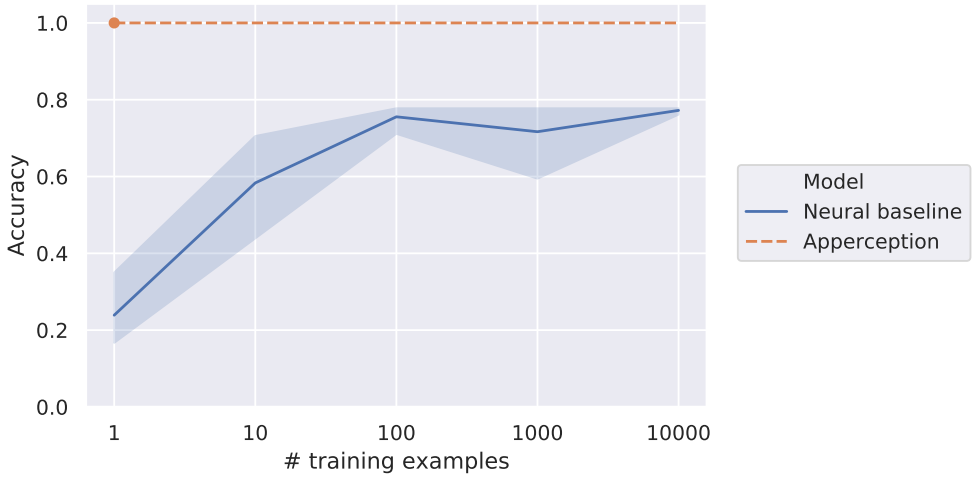

Fig. 1.7: The results on the Sokoban task. Apperception is trained on only a single example and the dashed line represents the apperception results on that single example. The neural baseline is trained on an increasing number of training examples. The shaded area is the 95% confidence interval on 10 runs with different random seeds.

timation classifier on its output. The classifier fits a Gaussian per class, trained on log-probabilities of independently sampled acceptable and unacceptable test states calculated over the Bernoulli distribution outputted by the model.

We trained the baseline with the Adam optimizer, varying the learning rate in $[0.05, 0.01, 0.005, 0.001]$, batch size in $[512, 1024]$ and executing each experiment 10

times. We selected the best set of hyperparameters by choosing the parameters with the best development set performance, and averaged the performance across 10 repetitions with different random seeds. During training, we annealed the temperature of the Gumbel-Softmax with an exponential decay, from 2.0 to 0.5 with a per-epoch decay of 0.0009.

As can be seen from Figure 1.7, the neural baseline is not able to correctly distinguish between acceptable and unacceptable next steps, neither from the single example, nor a large number of examples. However, as expected, the accuracy of the baseline increases with increasing size of the training set, though it hits a plateau without reaching the maximum. The Apperception Engine, on the other hand, is able to learn a fully accurate theory from a *single* carefully-chosen example.

1.4 Related work

Unsupervised learning from temporal sequences is central to statistics, engineering and the sciences, with countless applications. Most sequence models are probabilistic, allowing us to model the uncertainty and ambiguity arising from noisy measurements or partial observability. Traditionally, due to computational limitations, sequence models were informed by strong prior knowledge about particular domains and usually contained few tunable parameters, e.g (Kalman, 1960; Black and Scholes, 1973). The advent of deep learning methods and the availability of large scale data sets has revolutionized sequence modeling (Graves, 2013), by incorporating general learned function approximation to replace strong prior assumptions.

Probabilistic sequence models can broadly be divided into two classes (with some overlap): *auto-regressive* models and *latent variable* models. Auto-regressive models directly capture the temporal dependencies of observed variables by modeling the distribution of each time slice conditioned on the history of observations. This results in conceptually simple and fast model training procedures using maximum-likelihood estimation. At the same time, auto-regressive models for raw sensory data are usually large and therefore costly to evaluate at test time, hindering their application to e.g. planning problems with long-horizons where thousand of model evaluations are often necessary. Furthermore, they are usually not humanly interpretable and specifying prior domain knowledge is difficult.

Latent variable models can in principle address these shortcomings (Loehlin, 1987). They aim to capture the statistical dependencies of time series by positing latent variables that underlie the observed data (Ghahramani and Jordan, 1996). This latent structure is usually assumed to be simpler than the observed high-dimensional raw data, and can therefore in principle reduce the computational demands of long-rage predictions, therefore facilitating applications to e.g. reinforcement learning and planning (Buesing, Weber, Racaniere, Eslami, Rezende, Reichert, Viola, Besse, Gregor, Hassabis *et al.*, 2018). Furthermore, low-dimensional latent models are often used for finding simple, structured explanations in exploratory data analysis (Byron, Cunningham, Santhanam, Ryu, Shenoy and Sahani, 2009). Allowing for uncertainty over latent entities in order to capture multiple hypotheses of observed phenomena comes at the price of necessitating sophisticated approaches for fitting models to data. Most approaches either apply explicit probabilistic inference to determine the distributions

over latent variables from observations, or they rely on implicit or likelihood-free methods (Li, Monroe, Shi, Jean, Ritter and Jurafsky, 2017).

Modern latent variable sequence models mostly capture latent structure with contiguous variables, as this allows approximate model-fitting methods with gradient descent on parameters as a sub-routine (Kingma and Welling, 2013). However, predictions under these models are often subject to degrading fidelity with increasing prediction horizon. Although often inevitable to some degree, this effect is exacerbated by the inability of latent variables to capture discrete, or categorical structure, resulting in a "conceptual drift" in domains where the ground truth can be described by discrete concepts. Discrete (or mixed discrete-continuous (Johnson, Duvenaud, Wiltschko, Adams and Datta, 2016)) latent variable models can in principle alleviate this issues (Mnih and Rezende, 2016; van den Oord, Vinyals *et al.*, 2017), however, fitting these efficiently to data remains challenging (Maddison, Mnih and Teh, 2016).

1.5 Discussion

In this paper, we have shown how the Apperception Engine can be combined with a binary neural network in order to learn explicit causal theories from raw unprocessed sensory sequences. We demonstrated, in the *Sokoban* experiments, how this system is able to learn explicit, interpretable models from small amounts of noisy data. This is, to the best of our knowledge, the first system that can learn a provably correct dynamics model of a non-trivial game from raw unprocessed sensory input.

We pause to consider some limitations of our system in its current form. A fundamental limitation of the Apperception Engine is that it assumes that the underlying dynamics can be expressed as rules that operate on *discrete* concepts. While the system is capable of handling raw, noisy, continuous sensory input, it assumes that the *underlying dynamics* of the system can be represented by rules that operate on discrete concepts. There are many domains where the underlying dynamics are discrete while the surface output is noisy and continuous: Raven's progressive matrices, puzzle games, and Atari video games, for example. But our system will struggle in domains where the underlying dynamics are best modelled using continuous values, such as models of fluid dynamics. Here, the best our system could do is find a discrete model that crudely approximates the true continuous dynamics. Extending Datalog$^{\ni}$ to represent continuous change would be a substantial and ambitious project.

Another major limitation of our system is that it assumes that causal rules are strict, universal and exceptionless. There is no room in the current representation for defeasible causal rules (where normally, all other things being equal, a causes b) or nondeterministic causal rules (where a causes either b or c). In future work, we plan to implement nondeterministic causal rules.

A third limitation of our system is that it suffers from scaling issues, both in terms of memory and processing time. Finding a unified theory that explains the sensory input means searching through the space of logic programs. This is a huge and daunting task. For example, the Apperception Engine takes 5 Gigabytes of RAM and 48 hours to make sense of a single Sokoban trajectory consisting of 17 pixel arrays of size 20×20. This is, undeniably, a computationally expensive process.

Fig. 1.8: Using high-level conceptual information (in this case, the spelling of English words) to disambiguate low-level perceptual information. Here, there is a highly ambiguous symbol (in red) that is used for both the 'H' of 'THE' and for the 'A' of 'CAT'.

We would like to scale our approach up so that we can learn the dynamics of Atari games from raw pixels. But this will prove to be challenging, as games such as Pacman are harder than our Sokoban test-case in every dimension: it requires us to increase the number of pixels, the number of time-steps, the number of trajectories, the number of objects, and the complexity of the dynamics.

The dominant reason for our system's scaling difficulties is that it uses a maximising SAT solver to search through the space of logic programs. We encode the program synthesis problem as an ASP program, and find the simplest theory by encoding the theory size as a weak constraint. Finding an optimal solution to an ASP program with weak constraints is in Σ_2^P, but this complexity is a function of the number of ground atoms, and the number of ground atoms is exponential in the size of the Datalog$^{\neg}$ theory.

We are currently evaluating various different ways of improving the performance of our system so we can scale up to harder problems such as Atari, and are excited by the prospect of scaling up the Apperception Engine to the next level, so as to be able to induce robust causal models for Atari. But it will, we believe, require substantial further research.

1.6 Conclusion

When a human opens her eyes, she combines low-level perception (mapping raw unprocessed sensory input into objects and concepts) with high-level apperception (constructing a conceptual theory that makes sense of the stimulus).

Consider Figure 1.8 (Chalmers, French and Hofstadter, 1992). Here, the red letter is ambiguous between an 'A' and an 'H'. When we read the words, our high-level conceptual knowledge of the spelling of English words informs our low-level perceptual processing, so that we can effortlessly disambiguate the images.

We want our machines to do the same, combining low-level perception with high-level apperception. And we want the information to flow in *both directions*: as well as low-level perceptual information informing high-level conceptual theorising, we also want high-level conceptual considerations to inform low-level perceptual processing. The Apperception Engine is a proof of concept that such a system is, indeed, possible.

References

Black, Fischer and Scholes, Myron (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, **81**(3), 637–654.

Buesing, Lars, Weber, Theophane, Racaniere, Sebastien, Eslami, SM, Rezende, Danilo, Reichert, David P, Viola, Fabio, Besse, Frederic, Gregor, Karol, Hassabis, Demis et al. (2018). Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*.

Byron, M Yu, Cunningham, John P, Santhanam, Gopal, Ryu, Stephen I, Shenoy, Krishna V, and Sahani, Maneesh (2009). Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. In *Advances in Neural Information Processing Systems*, pp. 1881–1888.

Calimeri, Francesco, Faber, Wolfgang, Gebser, Martin, Ianni, Giovambattista, Kaminski, Roland, Krennwallner, Thomas, Leone, Nicola, Ricca, Francesco, and Schaub, Torsten (2012). Asp-core-2: Input language format. *ASP Standardization Working Group*.

Chalmers, David J, French, Robert M, and Hofstadter, Douglas R (1992). High-level perception, representation, and analogy: A critique of artificial intelligence methodology. *Journal of Experimental & Theoretical Artificial Intelligence*, **4**(3), 185–211.

Cheng, Chih-Hong, Nührenberg, Georg, Huang, Chung-Hao, and Ruess, Harald (2018). Verification of binarized neural networks via inter-neuron factoring. In *Working Conference on Verified Software: Theories, Tools, and Experiments*, pp. 279–290. Springer.

Evans, Richard, Hernandez-Orallo, Jose, Welbl, Johannes, Kohli, Pushmeet, and Sergot, Marek (2019). Making sense of sensory input. *arXiv preprint arXiv:1910.02227*.

Gebser, Martin, Kaminski, Roland, Kaufmann, Benjamin, and Schaub, Torsten (2014). Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*.

Ghahramani, Zoubin and Jordan, Michael I (1996). Factorial hidden Markov models. In *Advances in Neural Information Processing Systems*, pp. 472–478.

Graves, Alex (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Hubara, Itay, Courbariaux, Matthieu, Soudry, Daniel, El-Yaniv, Ran, and Bengio, Yoshua (2016). Binarized neural networks. In *Advances in Neural Information Processing Systems*, pp. 4107–4115.

Johnson, Matthew J, Duvenaud, David K, Wiltschko, Alex, Adams, Ryan P, and Datta, Sandeep R (2016). Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, pp. 2946–2954.

Kalman, Rudolph Emil (1960). A new approach to linear filtering and prediction

problems. *Journal of Basic Engineering*, **82**(1), 35–45.

Kant, Immanuel (1781). *Critique of Pure Reason*. Cambridge University Press.

Kim, Minje and Smaragdis, Paris (2016). Bitwise neural networks. *arXiv preprint arXiv:1601.06071*.

Kingma, Diederik P and Welling, Max (2013). Auto-encoding variational Bayes. *Proceedings of the International Conference on Learning Representations*.

Lee, Sau Dan and De Raedt, Luc (2004). Constraint based mining of first order sequences in seqlog. In *Database Support for Data Mining Applications*, pp. 154–173. Springer.

Leibniz, Gottfried Wilhelm (1765). *New Essays on Human Understanding*. Cambridge University Press.

Li, Jiwei, Monroe, Will, Shi, Tianlin, Jean, Sébastien, Ritter, Alan, and Jurafsky, Dan (2017). Adversarial learning for neural dialogue generation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Loehlin, John C (1987). *Latent Variable Models*. Lawrence Erlbaum Associates, Inc.

Longuenesse, Béatrice (1998). *Kant and the Capacity to Judge*. Princeton: Princeton UP.

Maddison, Chris J, Mnih, Andriy, and Teh, Yee Whye (2016). The concrete distribution: A continuous relaxation of discrete random variables. *Proceedings of the International Conference on Learning Representations*.

Mnih, Andriy and Rezende, Danilo J (2016). Variational inference for Monte Carlo objectives. *Proceedings of the International Conference on Machine Learning*.

Narodytska, Nina, Kasiviswanathan, Shiva, Ryzhyk, Leonid, Sagiv, Mooly, and Walsh, Toby (2018). Verifying properties of binarized deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Rastegari, Mohammad, Ordonez, Vicente, Redmon, Joseph, and Farhadi, Ali (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer.

Smith, Brian Cantwell (2019). *The Promise of Artificial Intelligence: Reckoning and Judgment*. MIT Press.

Tamaddoni-Nezhad, Alireza and Muggleton, Stephen (2009). The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. *Machine Learning*, **76**(1), 37–72.

van den Oord, Aaron, Vinyals, Oriol et al. (2017). Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pp. 6306–6315.