Learning Social Practices

Learning Social Practices

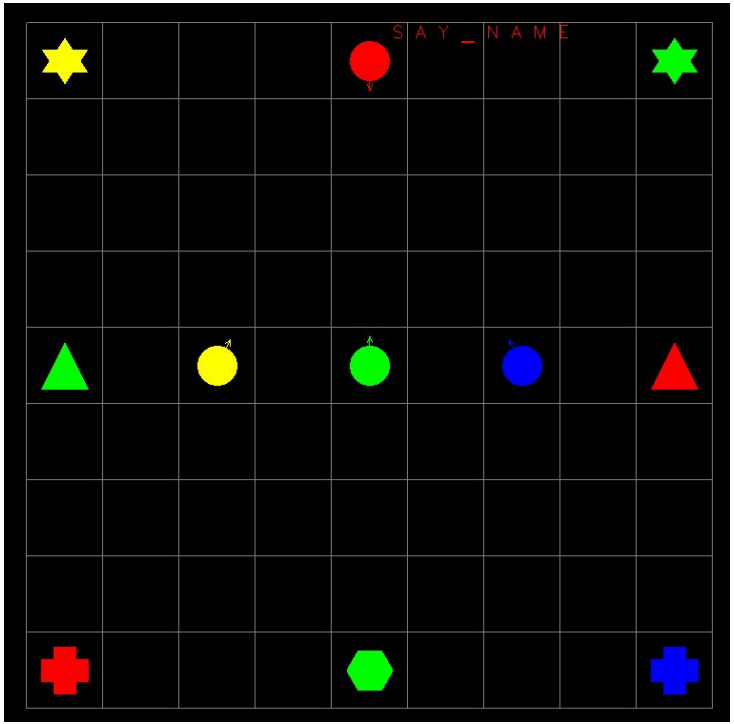
Children are able to learn a new social practice by observing a handful of traces of adult behaviour. How are they able to do this?

Apprenticeship Learning

Apprenticeship learning = learning by watching human-authored traces of competent / optimal behaviour

Applying apprenticeship learning to social practices = learning a social practice by watching traces of multi-agent behaviour

n00b



Apprenticeship Learning vs Reinforcement Learning

Turing recommended apprenticeship learning over reinforcement learning:

> *The use of punishments and rewards can at best be a part of the teaching process... By the time a child has learnt to repeat "Casablanca" he would probably feel **very sore indeed**, if the text could only be discovered by a "Twenty Questions" technique, every "NO" taking the form of a blow.*

Data-efficiency

Why data-efficiency is so important:

- Theoretical: a child can learn a new word from *one instance*. Carey et al showed children aged 3 learning a new word “chromium” from one exposure
- Practical: getting humans to produce behaviour traces is expensive

n00b’s approach

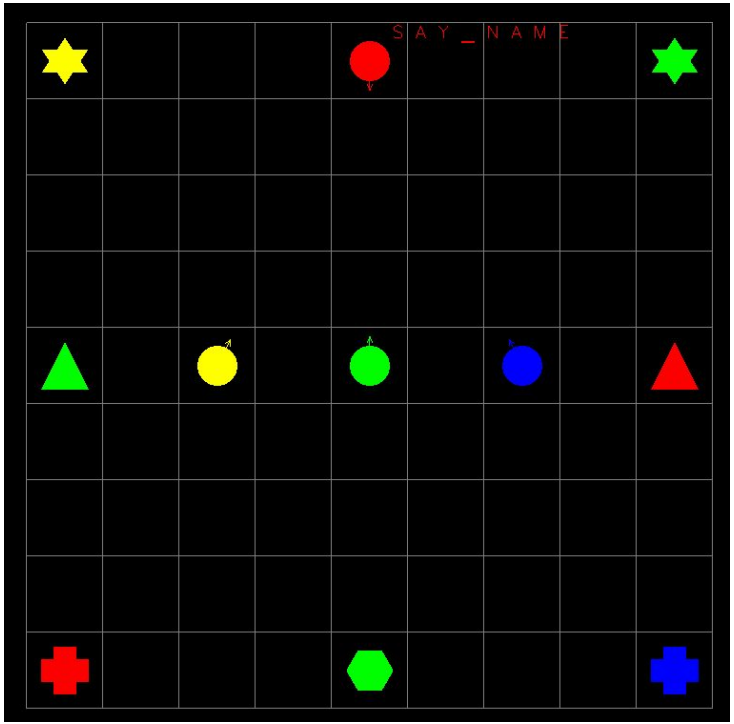
We apply apprenticeship learning to language games.

This system is data-efficient:

- One-shot learning
- Zero-shot learning
- Transfer learning

What makes this system data-efficient?

n00b



```
should(I, T2, S, name(S, Object1)) :-
        is(I, T1, perform(agent1, say(name))),
        is(I, T1, facing_object(M, Object1)),
        is_master(M),
        is_servant(S),
        less_than(T1, T2),
        instance(I),
        int(T2),
        agent(S),
        object(Object1).

is(I, T1, name(S, Object1)) :-
        is(I, T1, perform(S, say(ExtraWord1))),
        aux_name(ExtraWord1, Object1),
        instance(I),
        int(T1),
        agent(S),
        object(Object1).
```

Learning Explicit Rules of Practices

When humans learn a social practice, they are often unable to articulate precisely the norms of that practice. They may be able to effortlessly fit in, but they are not able to state precisely what they need to do to fit in.

Skilled practitioners of social practices typically have *procedural know-how*, rather than declarative know-that (Dreyfus 1992).

By contrast with humans, we want our computer agents to learn an explicit, readable model of a social practice, so we can interpret (and verify) their learnings.

```
should(I, T2, S, name(S, Object1)) :-
        is(I, T1, perform(agent1, say(name))),
        is(I, T1, facing_object(M, Object1)),
        is_master(M),
        is_servant(S),
        less_than(T1, T2),
        instance(I),
        int(T2),
        agent(S),
        object(Object1).

is(I, T1, name(S, Object1)) :-
        is(I, T1, perform(S, say(ExtraWord1))),
        aux_name(ExtraWord1, Object1),
        instance(I),
        int(T1),
        agent(S),
        object(Object1).
```

The Model

The Social Practice

A **social practice** is a collection of multi-agent behaviours with something in common. For example:

- A greeting
- A game
- A marriage
- The British legal system

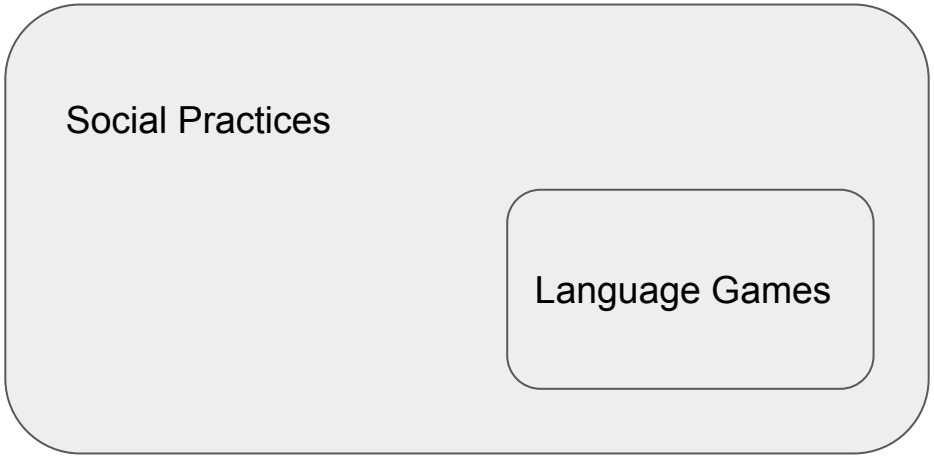
The Social Practice

The social world is a collection of concurrent social practices.

At any moment, many practices are simultaneously in play.

The set of practices is *continually evolving.*

A Language Game is a Type of Social Practice



Therefore: if we have a general system that can learn social practices, then we will have a system that can learn and understand language

Normative Systems

A **normative system** is a model of a social practice. It contains two types of rule:

- Normative: what the agents *should* do
- Descriptive: what *counts as* what

Example - greeting:

- Normative: when you see someone you haven't seen recently, you *should* greet them
- Descriptive: tipping your hat *counts as* greeting; saying "hello" *counts as* greeting...

Normative Systems

A **normative system** is a model of a social practice. It contains two types of rule:

- Normative: what the agents *should* do
- Descriptive: what *counts as* what

Example - conversation:

- Normative: if someone asks you a direct yes/no question, you *should* respond
- Descriptive: saying “yes" *counts as* responding to a direct yes/no question

Logic Programming

We represent a normative system by a set of rules in a defeasible multi-modal logic.

A Logic of Normative Systems

There are three fundamental predicates, using propositions reified as terms:

- *is(e, t, p)*: in episode *e*, at time *t*, proposition *p* is true
- *should(e, t, a, p)*: in episode *e,* at time *t,* agent *a* should make it the case that *p* is true (at time *t)*
- *will(e, t, a, p)*: in episode *e,* at time *t,* agent *a* decides to make it the case that *p* is true (at time *t*)

Example

A normative system is a set of pairs of rules: a normative rule and a corresponding descriptive rule.

```
should(E, T, A, greet(A, B)) :-
    agent(A),
    agent(B),
    is(E, T, at(A, X1, Y1)),
    is(E, T, at(B, X2, Y2)),
    adjacent(X1, Y1, X2, Y2).

is(E, T, greet(A, B)) :-
    is(E, T, perform(A, doff_hat_to(B))).
```

Interpreting the Physical World as a Social World

When applying a practice (a set of rules) to a set of bare physical facts, we derive a set of social facts. Consider the practice:

```
should(E, T, A, greet(A, B)) :-
    agent(A),
    agent(B),
    is(E, T, at(A, X1, Y1)),
    is(E, T, at(B, X2, Y2)),
    adjacent(X1, Y1, X2, Y2).

is(E, T, greet(A, B)) :-
    is(E, T, perform(A, doff_hat_to(B))).
```

Interpreting the Physical World as a Social World

When applying a practice (a set of rules) to a set of bare physical facts, we derive a set of social facts. For example, given:

```
is(e1, t1, at(alice, 3, 1))
is(e1, t1, at(bob, 5, 1))
is(e1, t2, at(alice, 4, 1))
is(e1, t2, at(bob, 5, 1))
is(e1, t3, perform(alice, doff_hat_to(bob)))
```

we derive:

```
should(e1, t2, alice, greet(alice, bob))
should(e1, t2, bob, greet(bob, alice))
is(e1, t3, greet(alice, bob))
```

Norm Satisfaction

A norm for *A* to make it the case that *P* at *T* is **satisfied** in episode ε if

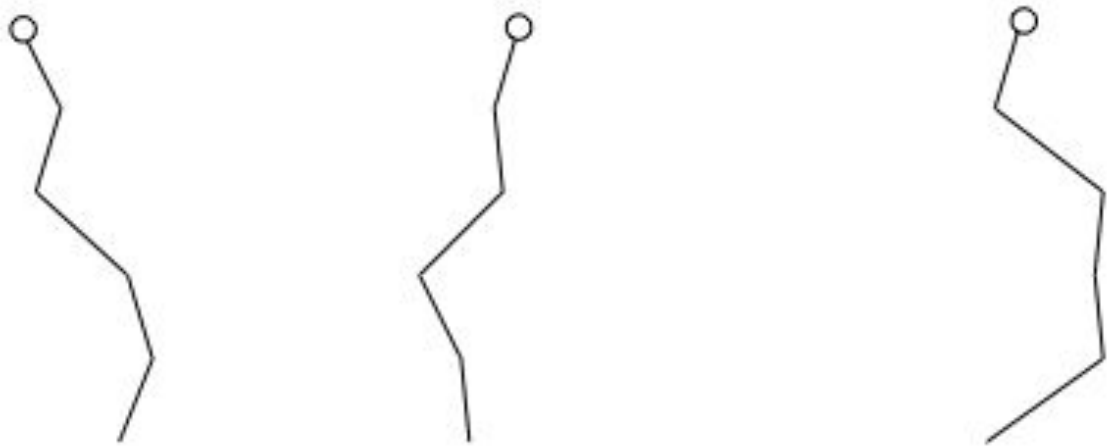
$$should(\varepsilon, T, A, P) \in \pi(\varepsilon) \text{ and } is(\varepsilon, T, P) \in \pi(\varepsilon)$$

Define the number σ of satisfied norms in episode ε according to social practice π as:

$$\sigma(\varepsilon, \pi) = |\{should(\varepsilon, T, A, P) \in \pi(\varepsilon) \mid is(\varepsilon, T, P) \in \pi(\varepsilon)\}|$$

Learning

The Original Trajectories



Adding Mutated Trajectories

Measuring How Well a Practice Explains Trajectories

Given a practice π and a suite *S* of episodes, together with their perturbed counterparts, define how well the practice π explains the episodes *S* as:

$$explains(\pi, S) = |\{\varepsilon \in S \mid \sigma(\varepsilon, \pi) > \sigma(\textsf{perturb}(\varepsilon), \pi)\}|$$

The learning task, then, is to find the practice that best explains the episodes:

$$\pi^* = \arg\max_{\pi} explains(\pi, S)$$

Algorithm 1: Learning a Social Practice

input : A suite S of episodes
output: A social practice (a logic program)

$A_{best}, s_{best}, \Pi_{best} \leftarrow \{\}, 0, \{\}$
$\tau \leftarrow$ `InitialTemplate` ()
repeat
  $\Pi \leftarrow$`GenerateClauses` (τ)
  $S^* \leftarrow \{\mathsf{perturb}(\varepsilon) \mid \varepsilon \in S\}$
  $P \leftarrow core \cup S \cup S^* \cup \Pi \cup opt$
  $A \leftarrow$ `Clingo` (P)
  $s \leftarrow explains(extract(A, \Pi), S)$
  if $s > s_{best}$ **then** $A_{best}, s_{best}, \Pi_{best} \leftarrow A, s, \Pi$
  $\tau \leftarrow$`NextTemplate` (τ)
until *time budget exceeded*
return $extract(A_{best}, \Pi_{best})$

A Logic for Practical Reasoning

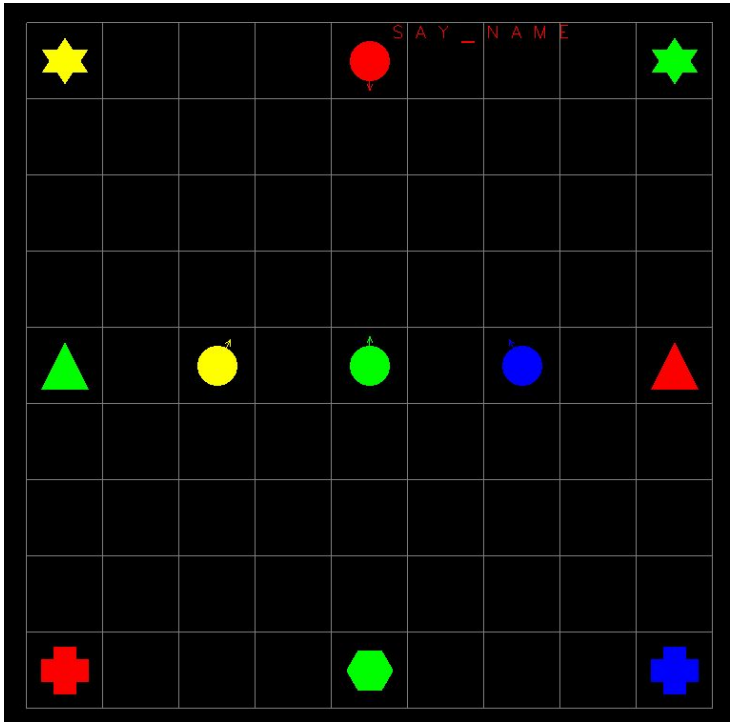
Agents decide to do what they believe they should do:

```
will(E, T, A, P) :-
    should(E, T, A, P),
    not -will(E, T, A, P).
```

An agent has direct control over certain propositions. For *some P*, he can will to make it true:

```
is(E, T, P) :-
    will(E, T, A, P),
    effector(P),
    not -is(E, T, P).
```

n00b

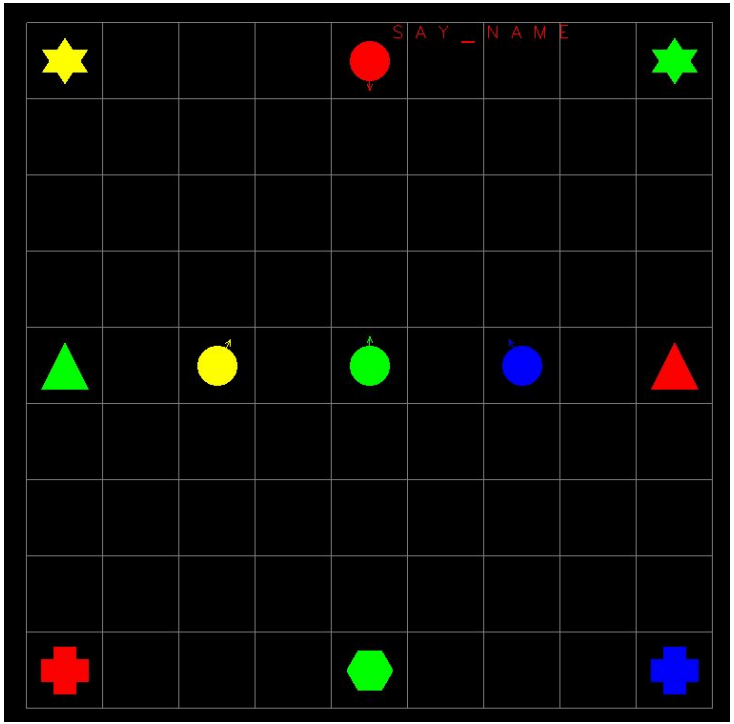


```
should(I, T2, S, name(S, Object1)) :-
        is(I, T1, perform(agent1, say(name))),
        is(I, T1, facing_object(M, Object1)),
        is_master(M),
        is_servant(S),
        less_than(T1, T2),
        instance(I),
        int(T2),
        agent(S),
        object(Object1).

is(I, T1, name(S, Object1)) :-
        is(I, T1, perform(S, say(ExtraWord1))),
        aux_name(ExtraWord1, Object1),
        instance(I),
        int(T1),
        agent(S),
        object(Object1).
```

```
aux_name_condition_1_1(immanuel, agent4).
aux_name_condition_2_1(gottlob, agent3).
aux_name_condition_3_1(ludwig, agent2).
aux_name_condition_4_1(ludwig, agent3).
```

n00b



```
should(I, T3, S, next(S, Object1)) :-
        is(I, T1, perform(M, say(ExtraWord1))),
        is_master(M),
        type(Object1, ExtraObject_type3),
        aux_name(ExtraWord1, S),
        is(I, T2, perform(M, say(ExtraWord2))),
        aux_type(ExtraWord2, ExtraObject_type3),
        is_master(M),
        is_servant(S),
        less_than(T1, T2),
        less_than(T2, T3),
        instance(I),
        int(T3),
        agent(S),
        object(Object1).

is(I, T1, next(S, Object1)) :-
        is(I, T1, next_to_object(Object1, S)),
        instance(I),
        int(T1),
        agent(S),
        object(Object1).
```

Experiments

Practice	simple	kNN(1)	kNN(5)	metagol	n00b
Name	0.49	0.66	0.78	1.0	1.0
Type	0.50	0.66	0.64	1.0	1.0
Next	0.49	0.68	0.85	-	1.0
Gaze	0.49	1.00	0.54	1.0	1.0
Color	0.51	0.48	0.66	1.0	1.0
Point	0.51	0.52	0.68	-	1.0
Question	0.50	0.66	0.66	-	1.0
Exists	0.50	0.83	0.67	-	1.0
ForAll	0.50	0.83	0.83	-	1.0
Unclean	0.49	0.62	0.61	-	1.0

What Makes n00b Data-Efficient?

Data-efficiency and prior knowledge

Data-efficiency requires prior knowledge.

To avoid parochial hand-engineered solutions, we need the most general *domain-independent* prior knowledge.

We want a general domain-independent invariant. For example:

- The conv-net: *objects retain their appearance as they move about*
- The recurrent net: *rules that are true at one time are also true at other times*

Data-efficiency and prior knowledge

The prior knowledge we inject is a general model of social practices and practical reasoning, encoded in a modal logic:

- A multi-modal logic distinguishing between **is**, **should**, and **will**
- General axioms relating **is**, **should**, and **will**
- A general template for a normative system as a pair of descriptive and normative rules:

```
should(E, T, A, P) :-
    ...
is(E, T, P) :-
    ...
```

The general template is a strong language bias.

How n00b Achieves Data-Efficiency

- Only considering rules with universally quantified variables
- Only considering *pairs* of (normative, descriptive) rules
- Giving the system a symbolic input (set of atoms) rather than raw pixels
- Giving the system a built-in physics model