Learning Explanatory Rules from Noisy Data

Richard Evans, Ed Grefenstette

DeepMind

Overview

Our system, ∂ILP, learns logic programs from examples.

∂ILP learns by back-propagation.

It is robust to noisy and ambiguous data.

Overview

1. Background
2. ∂ILP
3. Experiments

Learning Procedures from Examples

Given some input / output examples, learn
a **general procedure for transforming inputs into outputs**.

Learning Procedures from Examples

Given some input / output examples, learn
a **general procedure for transforming inputs into outputs**.

$$
\begin{aligned}
[\,] &\mapsto 0 \\
[2] &\mapsto 1 \\
[4, 3] &\mapsto 2 \\
[1, 2, 2] &\mapsto 3
\end{aligned}
$$

Learning Procedures from Examples

Given some input / output examples, learn
a **general procedure for transforming inputs into outputs**.

$$[[1]] \mapsto [[]]$$

$$[[4, 3]] \mapsto [[4]]$$

$$[[2, 3], [1]] \mapsto [[2], []]$$

$$[[1, 3, 2], [2, 4]] \mapsto [[1, 3], [2]]$$

Learning Procedures from Examples

We shall consider **three approaches**:

1. Symbolic program synthesis
2. Neural program induction
3. Neural program synthesis

Symbolic Program Synthesis (SPS)

Given some input/output examples, they produce an **explicit human-readable program** that, when evaluated on the inputs, produces the outputs.

They use a **symbolic search procedure** to find the program.

Symbolic Program Synthesis (SPS)

Input / Output Examples

$$[[1]] \mapsto [[]]$$
$$[[4,3]] \mapsto [[4]]$$
$$[[2,3],[1]] \mapsto [[2],[]]$$
$$[[1,3,2],[2,4]] \mapsto [[1,3],[2]]$$

Explicit Program

```
def remove_last(x):
  return [y[0:len(y)-1] for y in x]
```

Symbolic Program Synthesis (SPS)

Input / Output Examples

$$[[1]] \mapsto [[]]$$
$$[[4,3]] \mapsto [[4]]$$
$$[[2,3],[1]] \mapsto [[2],[]]$$
$$[[1,3,2],[2,4]] \mapsto [[1,3],[2]]$$

Explicit Program

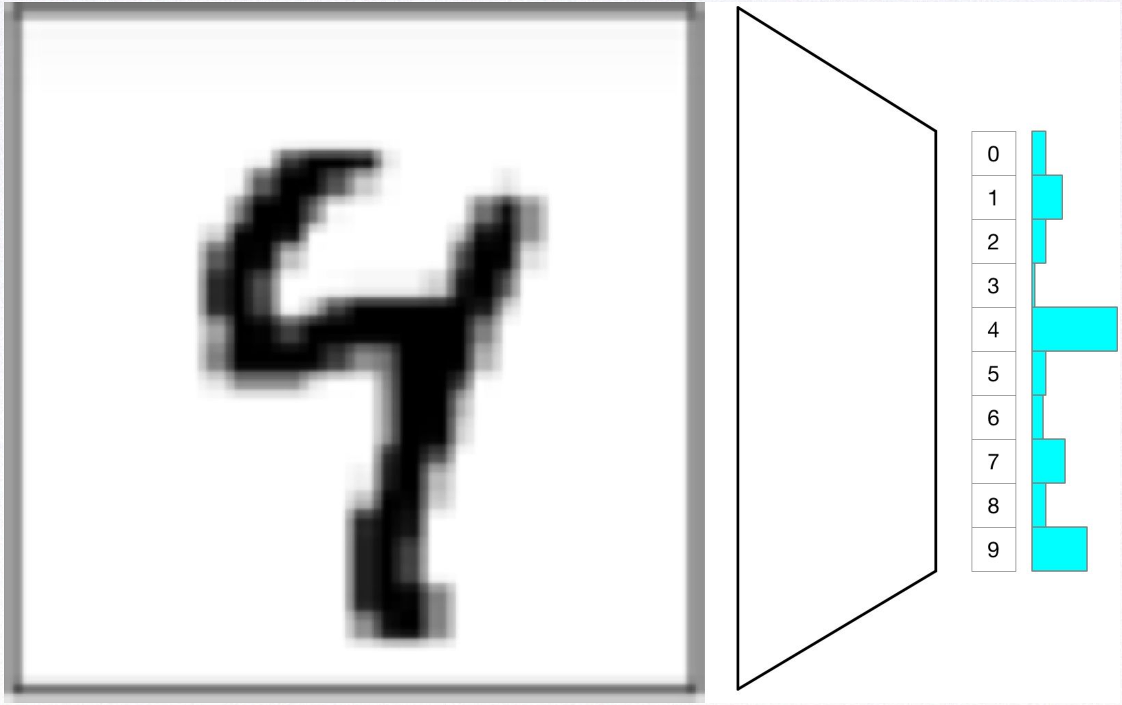
```
def remove_last(x):
  return [y[0:len(y)-1] for y in x]
```

Examples: MagicHaskeller, λ², Igor-2, Progol, Metagol

Symbolic Program Synthesis (SPS)

Data-efficient?	Yes
Interpretable?	Yes
Generalises outside training data?	Yes
Robust to mislabelled data?	Not very
Robust to ambiguous data?	No

Ambiguous Data



Neural Program Induction (NPI)

Given input/output pairs, a neural network learns a **procedure for mapping inputs to outputs**.

The network generates the output from the input directly, using a **latent representation of the program**.

Here, the general procedure is **implicit** in the weights of the model.

Neural Program Induction (NPI)

Examples:

Differentiable Neural Computers (Graves *et al.*, 2016)

Neural Stacks/Queues (Grefenstette *et al.*, 2015)

Learning to Infer Algorithms (Joulin & Mikolov, 2015)

Neural Programmer-Interpreters (Reed and de Freitas, 2015)

Neural GPUs (Kaiser and Sutskever, 2015)

Neural Program Induction (NPI)

Data-efficient?	Not very
Interpretable?	No
Generalises outside training data?	Sometimes
Robust to mislabelled data?	Yes
Robust to ambiguous data?	Yes

The Best of Both Worlds?

	SPS	NPI	Ideally
Data-efficient?	Yes	Not always	Yes
Interpretable?	Yes	No	Yes
Generalises outside training data?	Yes	Not always	Yes
Robust to mislabelled data?	Not very	Yes	Yes
Robust to ambiguous data?	No	Yes	Yes

Neural Program Synthesis (NPS)

Given some input/output examples, produce an **explicit human-readable program** that, when evaluated on the inputs, produces the outputs.

Use an **optimisation procedure** (e.g. gradient descent) to find the program.

Neural Program Synthesis (NPS)

Given some input/output examples, produce an **explicit human-readable program** that, when evaluated on the inputs, produces the outputs.

Use an **optimisation procedure** (e.g. gradient descent) to find the program.

Examples: ∂ILP, RobustFill, Differentiable Forth, End-to-End Differentiable Proving

The Three Approaches

	Procedure is implicit	*Procedure is explicit*
Symbolic search		Symbolic Program Synthesis
Optimisation procedure	Neural Program Induction	Neural Program Synthesis

The Three Approaches

	SPS	NPI	NPS
Data-efficient?	Yes	Not always	Yes
Interpretable?	Yes	No	Yes
Generalises outside training data?	Yes	Not always	Yes
Robust to mislabelled data?	No	Yes	Yes
Robust to ambiguous data?	No	Yes	Yes

∂ILP

∂ILP uses a differentiable model of forward chaining inference.

The weights represent a probability distribution over clauses.

We use SGD to minimise the log-loss.

We extract a readable program from the weights.

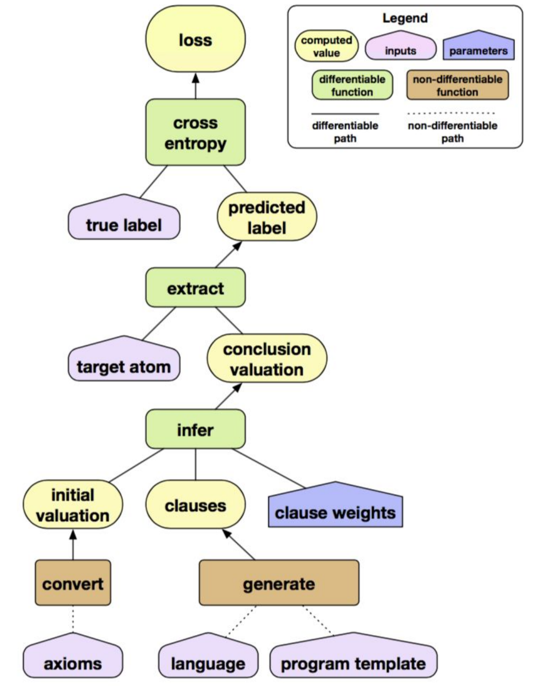


Figure 1: The ∂ILP Architecture.

∂ILP

A **valuation** is a vector in $[0,1]^n$

It maps each of *n* ground atoms to [0,1].

A valuation represents how likely it is that each of the ground atoms is true.

G	$\mathbf{a}_0$
$p(a)$	0.0
$p(b)$	0.0
$q(a)$	0.1
$q(b)$	0.3
$\perp$	0.0

∂ILP

Each clause *c* is compiled into a function on valuations:

$$F_c : [0,1]^n \rightarrow [0,1]^n$$

For example:

$$p(X) \leftarrow q(X)$$

G	$\mathbf{a}_0$	$\mathcal{F}_c(\mathbf{a_0})$
$p(a)$	0.0	0.1
$p(b)$	0.0	0.3
$q(a)$	0.1	0.0
$q(b)$	0.3	0.0
$\perp$	0.0	0.0

∂ILP

We combine the clauses' valuations using a weighted sum:

$$b_t = \sum_c F_c(a_t) \frac{e^{W[c]}}{\sum_{c'} e^{W[c']}}$$

We amalgamate the previous valuation with the new clauses' valuation:

$$a_{t+1} = a_t + b_t - a_t \cdot b_t$$

We unroll the network for *T* steps of forward-chaining inference, generating:

$$a_0, a_1, a_2, a_3, ..., a_T$$

∂ILP

∂ILP uses a differentiable model of forward chaining inference.

The weights represent a probability distribution over clauses.

We use SGD to minimise the log-loss.

We extract a readable program from the weights.

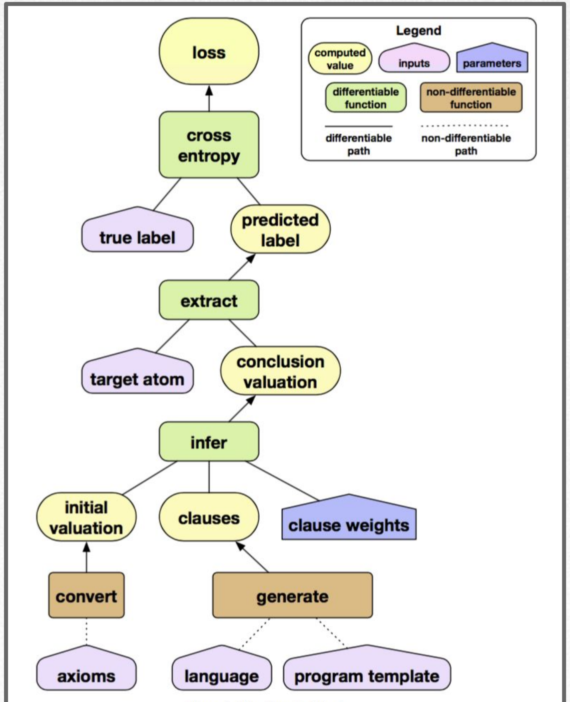


Figure 1: The ∂ILP Architecture.

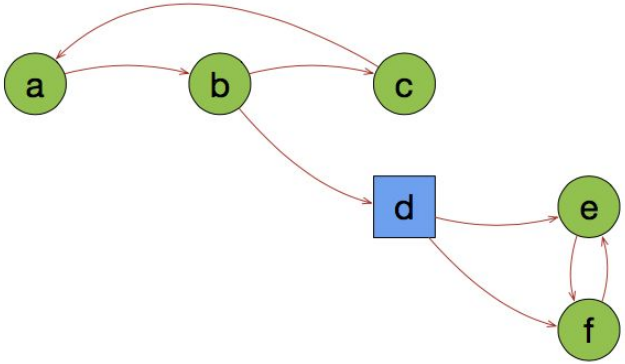
∂ILP Experiments

DeepMind

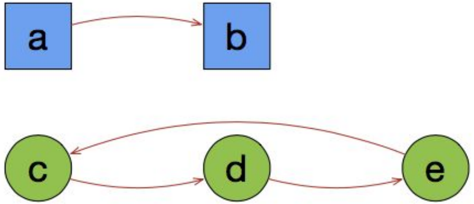
Domain	Task	$\|P_i\|$	Recursive	Metagol Performance	∂ILP Performance
Arithmetic	Predecessor	1	No	✓	✓
Arithmetic	Even / odd	2	Yes	✓	✓
Arithmetic	Even / succ2	2	Yes	✓	✓
Arithmetic	Less than	1	Yes	✓	✓
Arithmetic	Fizz	3	Yes	✓	✓
Arithmetic	Buzz	2	Yes	✓	✓
Lists	Member	1	Yes	✓	✓
Lists	Length	2	Yes	✓	✓
Family Tree	Son	2	No	✓	✓
Family Tree	Grandparent	2	No	✓	✓
Family Tree	Husband	2	No	✓	✓
Family Tree	Uncle	2	No	✓	✓
Family Tree	Relatedness	1	No	×	✓
Family Tree	Father	1	No	✓	✓
Graphs	Undirected Edge	1	No	✓	✓
Graphs	Adjacent to Red	2	No	✓	✓
Graphs	Two Children	2	No	✓	✓
Graphs	Graph Colouring	2	Yes	✓	✓
Graphs	Connectedness	1	Yes	×	✓
Graphs	Cyclic	2	Yes	×	✓

Table 2: A Comparison Between ∂ILP and Metagol on 20 Symbolic Tasks

Example Task: Graph Cyclicity



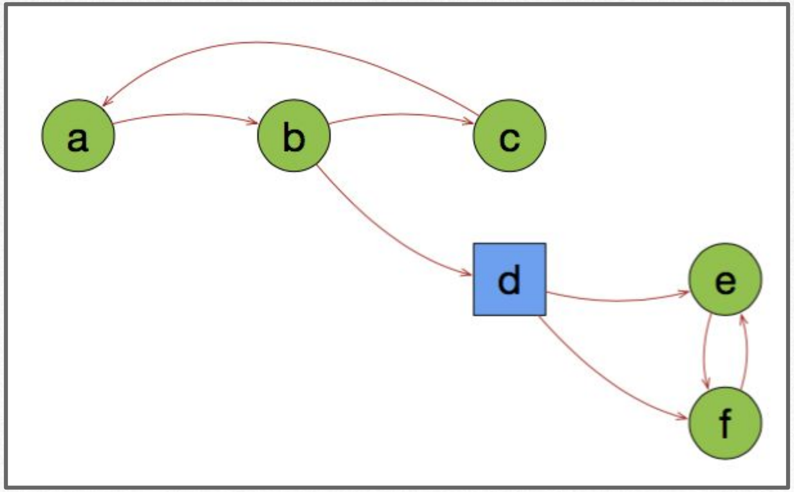
(a) First Training Triple



(b) Second Training Triple

(c) Validation Triple

Example Task: Graph Cyclicity




```
cycle(X) ← pred(X, X).

pred(X, Y) ← edge(X, Y).

pred(X, Y) ← edge(X, Z), pred(Z, Y)
```

Example: Fizz-Buzz

```
1 ↦ 1
2 ↦ 2
3 ↦ Fizz
4 ↦ 4
5 ↦ Buzz
6 ↦ Fizz
7 ↦ 7
8 ↦ 8
9 ↦ Fizz
10 ↦ Buzz
```

```
11 ↦ 11
12 ↦ Fizz
13 ↦ 13
14 ↦ 14
15 ↦ Fizz+Buzz
16 ↦ 16
17 ↦ 17
18 ↦ Fizz
19 ↦ 19
20 ↦ Buzz
```

Example: Fizz

```
fizz(X) ← zero(X).

fizz(X) ← fizz(Y), pred1(Y, X).

pred1(X, Y) ← succ(X, Z), pred2(Z, Y).

pred2(X, Y) ← succ(X, Z), succ(Z, Y).
```

Example: Fizz

```
fizz(X) ← zero(X).

fizz(X) ← fizz(Y), pred1(Y, X).

pred1(X, Y) ← succ(X, Z), pred2(Z, Y).

pred2(X, Y) ← succ(X, Z), succ(Z, Y).
```

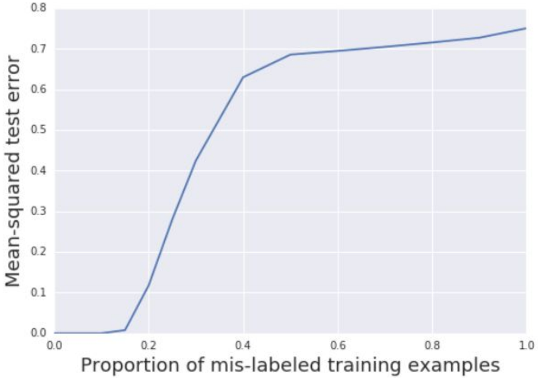
Example: Buzz

```
buzz(X) ← zero(X).

buzz(X) ← buzz(Y), pred3(Y, X).

pred3(X, Y) ← pred1(X, Z), pred2(Z, Y).
```

```
pred1(X, Y) ← succ(X, Z), pred2(Z, Y).

pred2(X, Y) ← succ(X, Z), succ(Z, Y).
```

Mis-labelled Data

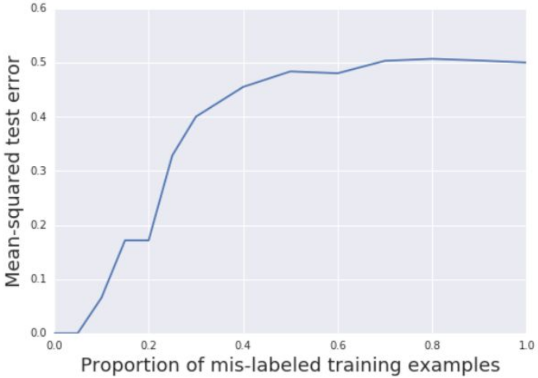
- If Symbolic Program Synthesis is given a single mis-labelled piece of training data, it **fails catastrophically**.
- We tested ∂ILP with mis-labelled data.
- We mis-labelled a certain proportion ρ of the training examples.
- We ran experiments for different values of $\rho = 0.0, 0.1, 0.2, 0.3, \ldots$



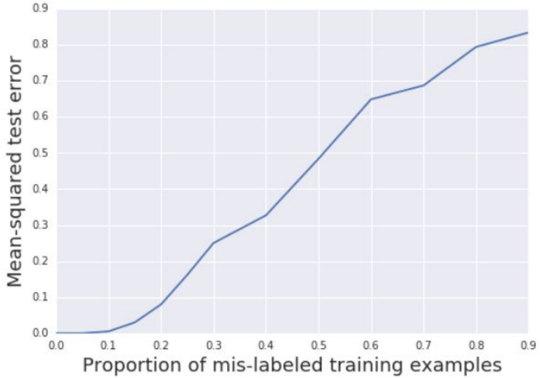
(a) Predecessor

(b) Less-Than

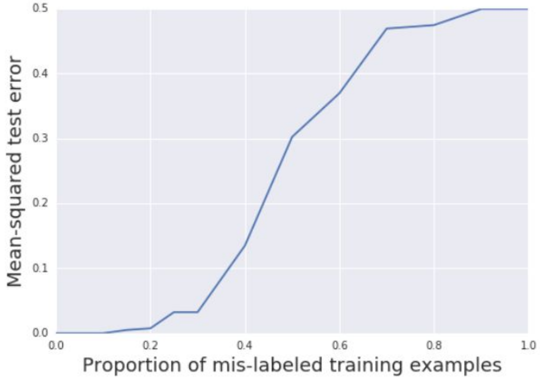
(c) List: Member



(d) Family Tree: Son



(e) Graph: Connectedness



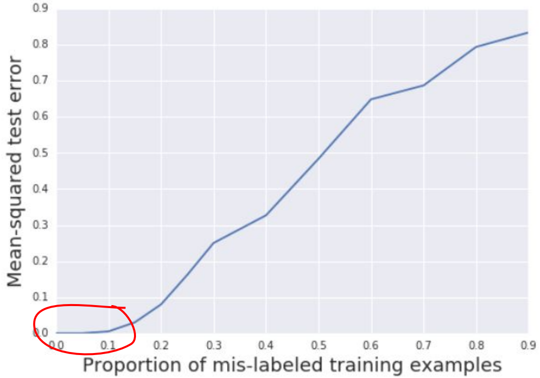
(f) Graph: Undirected Edge

(a) Predecessor

(b) Less-Than

(c) List: Member

(d) Family Tree: Son



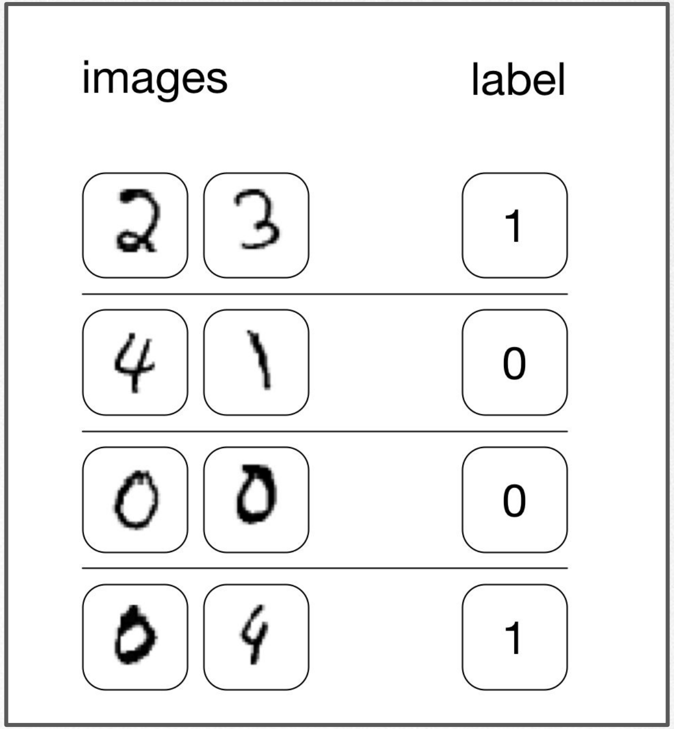
(e) Graph: Connectedness

(f) Graph: Undirected Edge

Example: Learning Rules from Ambiguous Data

Your system observes:

- a pair of images
- a label indicating whether the left image is *less than* the right image



Example: Learning Rules from Ambiguous Data

Your system observes:

- a pair of images
- a label indicating whether the left image is *less than* the right image

Two forms of generalisation:
It must decide if the relation holds for held-out images, and also *held-out pairs of digits*.

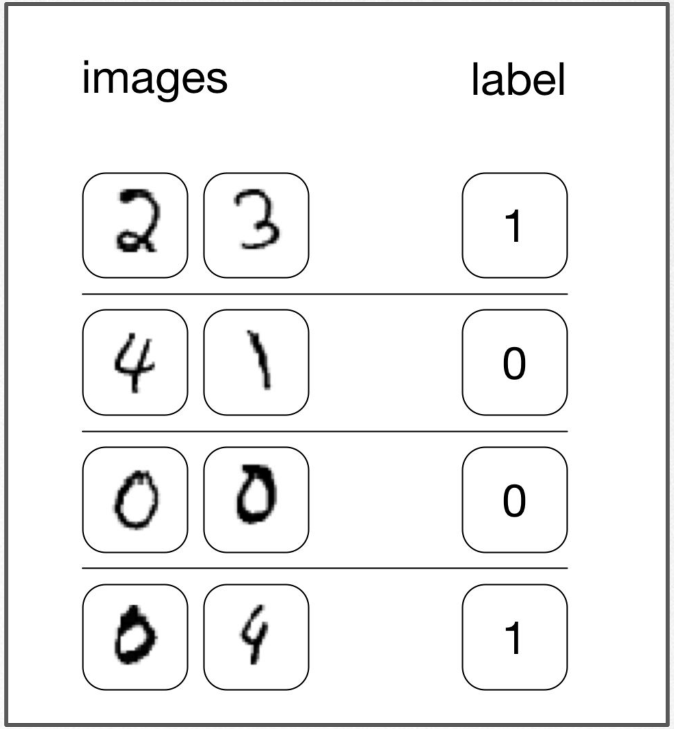
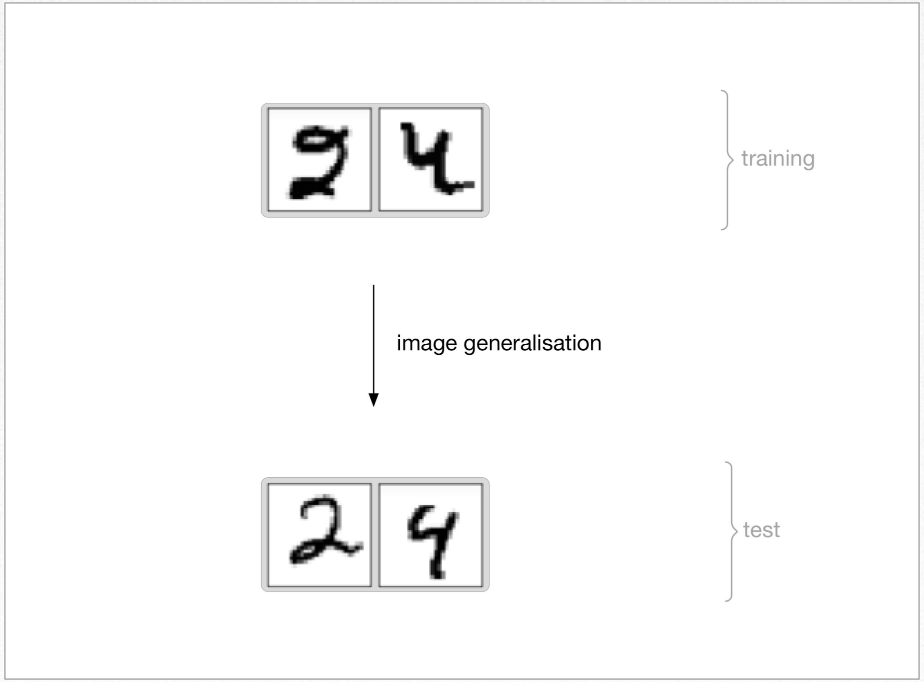
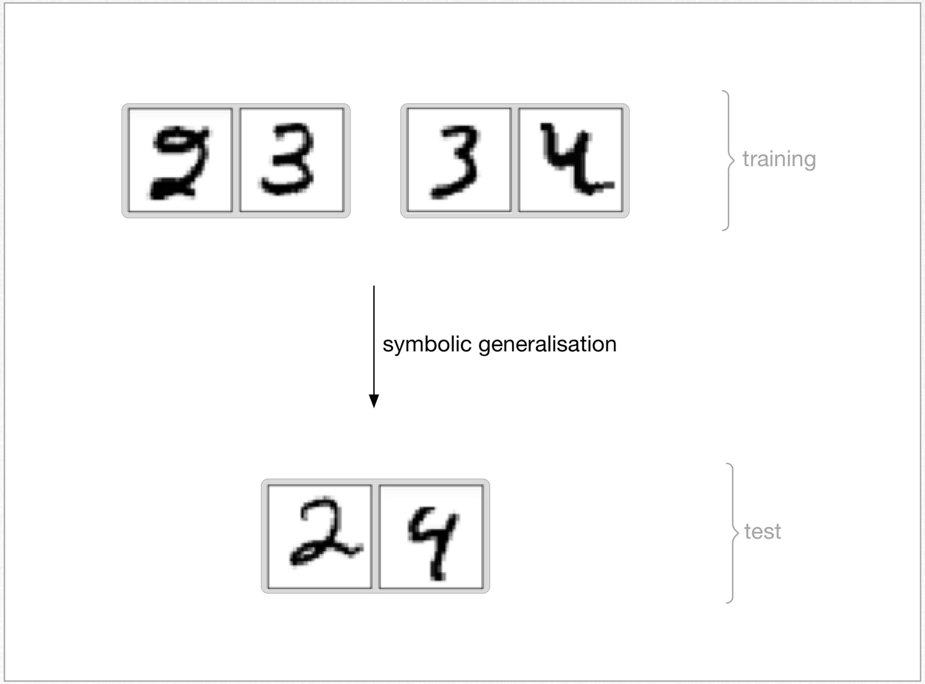


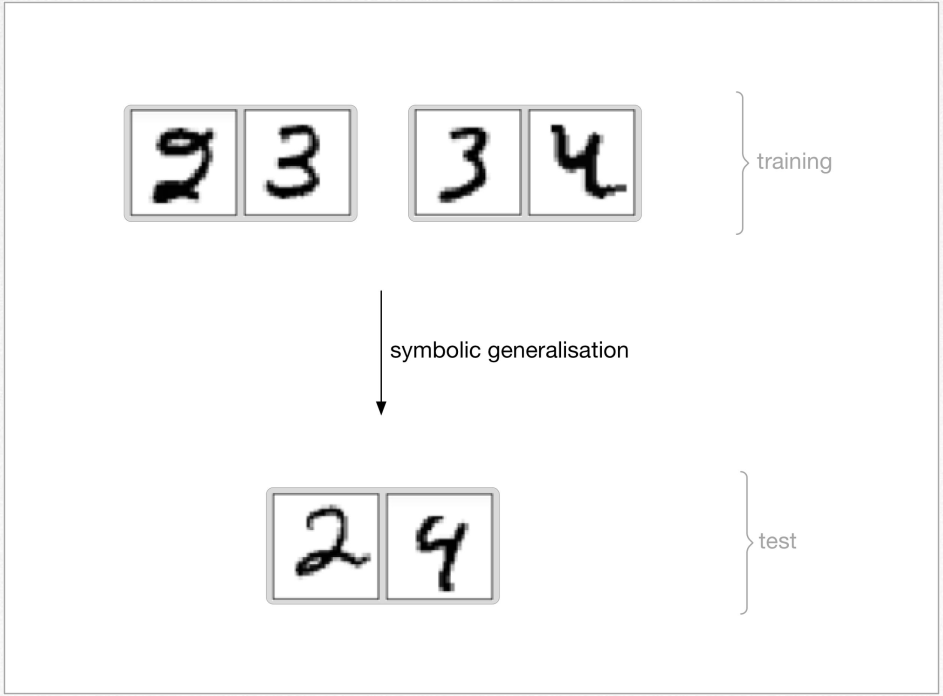
Image Generalisation



Symbolic Generalisation



Symbolic Generalisation



NB it has never seen any examples of 2 < 4 in training

Symbolic Generalisation

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7	0 < 8	0 < 9
		1 < 2	1 < 3	1 < 4	1 < 5	1 < 6	1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7	2 < 8	2 < 9
				3 < 4	3 < 5	3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	4 < 9
						5 < 6	5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	7 < 9
									8 < 9

Symbolic Generalisation

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7	0 < 8	0 < 9
		1 < 2	1 < 3	1 < 4	1 < 5	1 < 6	1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7	2 < 8	2 < 9
				3 < 4	3 < 5	3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	4 < 9
						5 < 6	5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	7 < 9
									8 < 9

Symbolic Generalisation

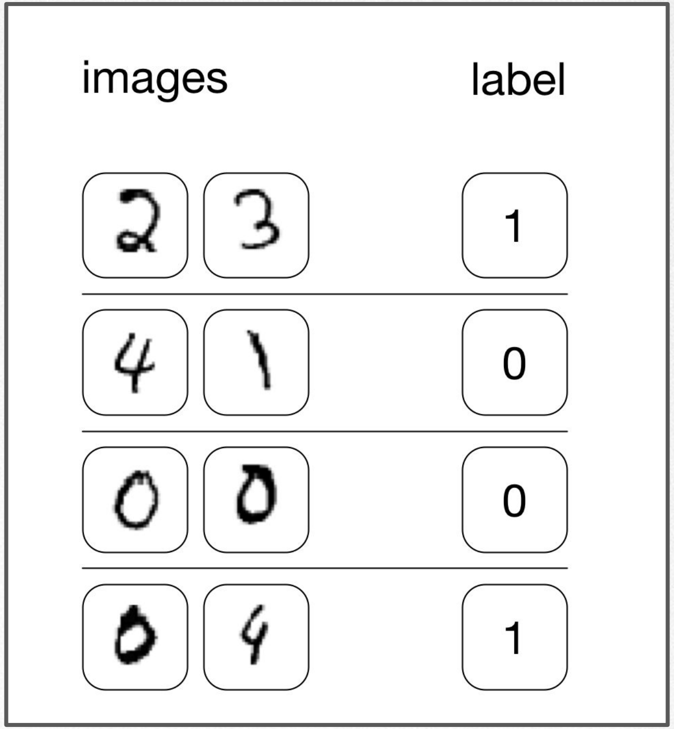
	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7	0 < 8	0 < 9
		1 < 2	1 < 3	1 < 4	1 < 5		1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7		2 < 9
				3 < 4		3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	4 < 9
						5 < 6	5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	
									8 < 9

Example: Less Than on MNIST Images

Your system observes:

- a pair of images
- a label indicating whether the left image is *less than* the right image

Two forms of generalisation:
It must decide if the relation holds for held-out images, and also *held-out pairs of digits*.



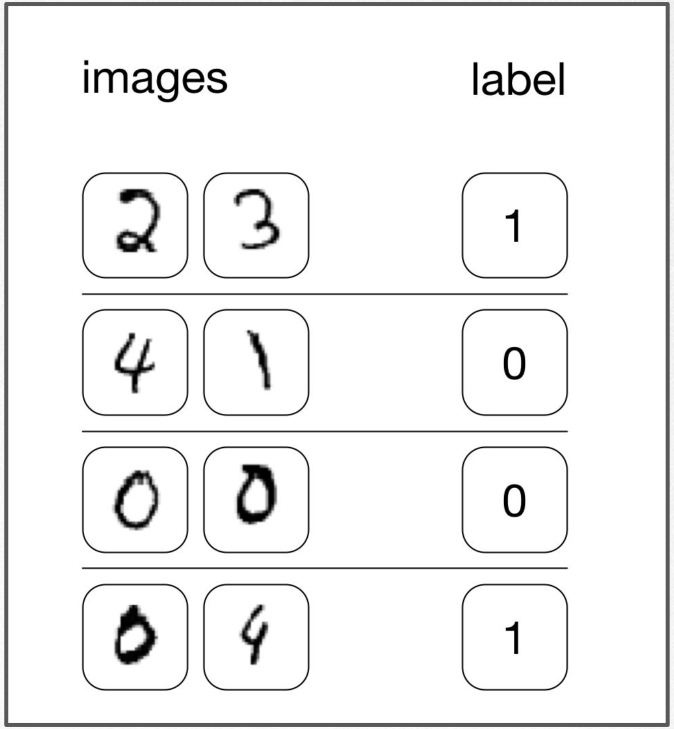
MLP Baseline

We created a baseline MLP to solve this task.

The output of the conv-net for the two images is a vector of (20) logits.

We added a hidden layer, produced a single output, and trained on cross-entropy loss.

The MLP baseline can solve this task easily.



Example: Less Than

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7	0 < 8	0 < 9
		1 < 2	1 < 3	1 < 4	1 < 5	1 < 6	1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7	2 < 8	2 < 9
				3 < 4	3 < 5	3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	4 < 9
						5 < 6	5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	7 < 9
									8 < 9

Example: Less Than

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7	0 < 8	0 < 9
		1 < 2	1 < 3	1 < 4	1 < 5	1 < 6	1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7	2 < 8	2 < 9
				3 < 4	3 < 5	3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	4 < 9
						5 < 6	5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	7 < 9
									8 < 9

Example: Less Than

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7	0 < 8	0 < 9
		1 < 2	1 < 3	1 < 4	1 < 5		1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7		2 < 9
				3 < 4		3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	4 < 9
						5 < 6	5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	
									8 < 9

Example: Less Than

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7	0 < 8	0 < 9
		1 < 2	1 < 3	1 < 4	1 < 5		1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7		2 < 9
				3 < 4		3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	4 < 9
						5 < 6	5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	
									8 < 9

Example: Less Than

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7		0 < 9
		1 < 2		1 < 4	1 < 5		1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7		2 < 9
				3 < 4		3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	
							5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	
									8 < 9

Example: Less Than

	0 < 1	0 < 2	0 < 3	0 < 4	0 < 5	0 < 6	0 < 7		0 < 9
		1 < 2		1 < 4	1 < 5		1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7		2 < 9
				3 < 4		3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	
							5 < 7	5 < 8	5 < 9
							6 < 7	6 < 8	6 < 9
								7 < 8	
									8 < 9

Example: Less Than

	0 < 1	0 < 2		0 < 4	0 < 5	0 < 6	0 < 7		0 < 9
		1 < 2		1 < 4			1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7		
				3 < 4		3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	
							5 < 7	5 < 8	5 < 9
									6 < 9
								7 < 8	
									8 < 9

Example: Less Than

	0 < 1	0 < 2		0 < 4	0 < 5	0 < 6	0 < 7		0 < 9
		1 < 2		1 < 4			1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5	2 < 6	2 < 7		
				3 < 4		3 < 6	3 < 7	3 < 8	3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	
							5 < 7	5 < 8	5 < 9
									6 < 9
								7 < 8	
									8 < 9

Example: Less Than

	0 < 1			0 < 4	0 < 5	0 < 6	0 < 7		0 < 9
		1 < 2		1 < 4			1 < 7	1 < 8	1 < 9
			2 < 3	2 < 4	2 < 5		2 < 7		
				3 < 4		3 < 6			3 < 9
					4 < 5	4 < 6	4 < 7	4 < 8	
							5 < 7	5 < 8	5 < 9
									6 < 9
								7 < 8	

∂ILP Learning Less-Than

We made a slight modification to our original architecture:

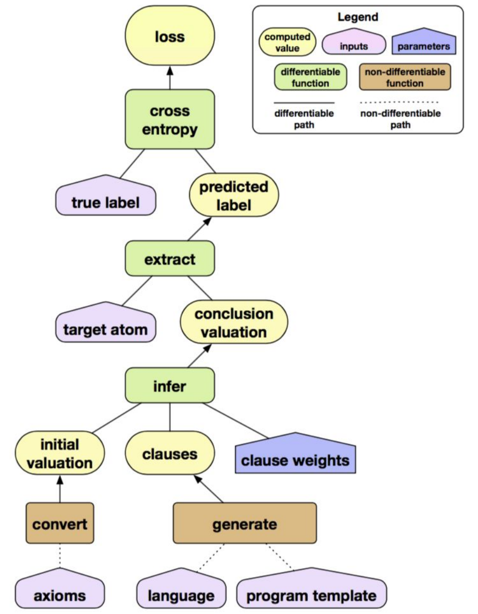


Figure 1: The ∂ILP Architecture.

∂ILP Learning Less-Than

We pre-trained a conv-net to recognise MNIST digits.

We convert the logits of the conv-net into a probability distribution over logical atoms.

Our model is able to solve this task.

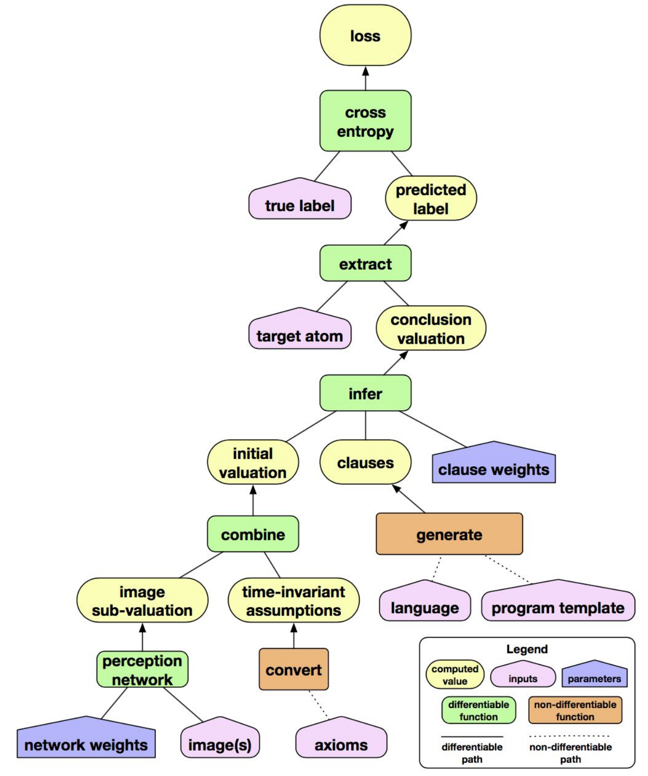


Figure 5: ∂ILP from Raw Pixel Images

∂ILP Learning Less-Than

```
target() ← image2(X), pred1(X)

pred1(X) ← image1(Y), pred2(Y, X)

pred2(X, Y) ← succ(X, Y)

pred2(X, Y) ← pred2(Z, Y), pred2(X, Z)
```

Comparing ∂ILP with the Baseline

Comparing ∂ILP with the Baseline

Conclusion

∂ILP aims to combine the advantages of Symbolic Program Synthesis with the advantages of Neural Program Induction:

- It has low *sample complexity*
- It can learn *interpretable* and *general* rules
- It is robust to *mislabelled* data
- It can handle *ambiguous* input
- It can be integrated and trained jointly within larger neural systems/agents