What can abstraction do for program induction

Sebastijan Dumančić

MIT, December 2020

Which task is easier to solve?

$27 + 15 + 9$	CIX + L + IX

How we represent a problem matters

Inventing abstractions: humans excel at adapting representations to their needs

Effective communication

> Penguin vs 'medium sized bird with wings that cannot fly'

Effective problem solving

> 5 x 2 vs 2 + 2 + 2 + 2 + 2

How we represent a problem matters

Inventing abstractions: humans excel at adapting representations to their needs

Effective communication

> Penguin vs 'medium sized bird with wings that cannot fly'

Effective problem solving

> 5 x 2 vs 2 + 2 + 2 + 2 + 2

How do we discover useful symbolic abstractions?

Why does it matter?

Why does it matter?

Why does it matter?

Where does the abstraction fit?

Where does the abstraction fit?

Abstraction before learning

Where does the abstraction fit?

> Discovering abstractions before learning

> Discovering abstractions after/within learning

> Creating a smoother landscape for program induction

> Unifying high-level reasoning with perception

Discovering abstractions via symmetry

Similar objects create categories

versus

Similarity in general knowledge representations?

Dumancic and Blockeel; MLJ 2017

Similarity in general knowledge representations?

Dumancic and Blockeel; MLJ 2017

Similarity in general knowledge representations?

Dumancic and Blockeel; MLJ 2017

Dumancic and Blockeel; IJCAI 2017

Dumancic and Blockeel; IJCAI 2017

Dumancic and Blockeel; IJCAI 2017

Symmetry is a useful proxy for abstractions

Learning from abstracted representation leads to smaller errors

Discovering abstractions via compression

How can we express X more efficiently?

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Discovering abstractions via compression

How can we express *X* more efficiently?

auto-encoders: a versatile (abstraction) learning block

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Discovering abstractions via compression

Auto-encoding logic programs: auto-encoders with logic programs as a computational engine

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

1. extract candidate abstractions

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Learning as constraint solving

choose the abstractions and reconstruction primitives

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

Does learning from ALP-created latent representation help with learning?

Yes!

Dumancic, Meert, Guns, Blockeel; IJCAI 2019

> Discovering abstractions before learning

> Discovering abstractions after/within learning

> Creating a smoother landscape for program induction

> Unifying high-level reasoning with perception

How do we effectively use knowledge?

2

Accretion, Tuning, and Restructuring: Three Modes of Learning

David E. Rumelhart
Donald A. Norman

University of California, San Diego

Learning through accretion
daily accumulation of information

Learning through tuning […]
changes in the very categories

"Learning through restructuring […] new structures are devised […]
new organisation on that already stored.

Dumancic, Guns, Cropper (Oxford); AAAI 2021

How do we effectively use knowledge?

Learning through accretion
daily accumulation of information

Learning through tuning […]
changes in the very categories

"Learning through restructuring […] new structures are devised […]
new organisation on that already stored.

Dumancic, Guns, Cropper (Oxford); AAAI 2021

The challenge:

How to restructure knowledge to make a better use of it?

The challenge:

How to restructure knowledge to make a better use of it?

Programs store knowledge explicitly!

```
def lego_pillar(X):
    place( , X)
    Y = X + 1
    for _ in range(3):
        place( , Y)
    place( , X)
```

The knowledge refactoring problem

Agent's knowledge:

The knowledge refactoring problem

Agent's knowledge:

Knowledge refactoring:

How can we achieve the same goal but more efficiently?

The knowledge refactoring problem

Agent's knowledge:

Knowledge refactoring:

Picking the right abstractions for the job

Find a (syntactically) equivalent program that is:

- smaller
- (- less redundant)

Transforming programs: folding and unfolding

```
def lego_pillar(X):
    place( , X)
    Y = X + 1
    for _ in range(3):
        place( , Y)
    place( , X)
```

Transforming programs: folding and unfolding

```
def lego_pillar(X):
    place( ▭ , X)
    Y = X + 1
    for _ in range(3):
        place( □ , Y)
    place( ▭ , X)
```

Folding →

```
def lego_pillar(X):
    place( ▭ , X)
    Y = X + 1
    place_vertical(Y)
    place( ▭ , X)

def place_vertical(Y):
    for _ in range(3):
        place( □ , Y)
```

Transforming programs: folding and unfolding

```
def lego_pillar(X):
    place( ▭ , X)
    Y = X + 1
    for _ in range(3):
        place( □ , Y)
    place( ▭ , X)
```

Folding →

← Unfolding

```
def lego_pillar(X):
    place( ▭ , X)
    Y = X + 1
    place_vertical(Y)
    place( ▭ , X)

def place_vertical(Y):
    for _ in range(3):
        place( □ , Y)
```

Finding and choosing the abstractions

Initial program

Finding and choosing the abstractions

Initial program

Unfolded program

Finding and choosing the abstractions

Finding and choosing the abstractions

Finding and choosing the abstractions

Finding and choosing the abstractions

Finding and choosing the abstractions

choose a subset of
abstractions

Finding and choosing the abstractions

choose a subset of
abstractions

… so that one of these
is possible to construct

Finding and choosing the abstractions

choose a subset of abstractions

… so that one of these is possible to construct

Objective:

minimise program size and redundancy

Does refactoring help to learn more efficiently?

Lifelong learning

learning to solve a series of tasks

Cropper: *Learning programs through play*, IJCAI 2019

Does refactoring help to learn more efficiently?

Phase I

"Playground"

building agent's knowledge

Task 1 Task 2 Task 3 Task 4 Task 5

Phase II

Lifelong learning

learning to solve a series of tasks

Task 1 Task 2 Task 3 Task 4 Task 5

Cropper: *Learning programs through play*, IJCAI 2019

Does refactoring help to learn more efficiently?

Phase I

"Playground"

building agent's knowledge

Phase II

Lifelong learning

learning to solve a series of tasks

Cropper: *Learning programs through play*, IJCAI 2019

Does refactoring help to learn more efficiently?

Phase I

"Playground"

building agent's knowledge

Phase II

Lifelong learning

learning to solve a series of tasks

Cropper: *Learning programs through play*, IJCAI 2019

Does refactoring help to learn more efficiently?

Phase I

"Playground"

building agent's knowledge

Refactor Refactor

200 - 4000 tasks

Phase II

Lifelong learning

learning to solve a series of tasks

Cropper: *Learning programs through play*, IJCAI 2019

String transformations
Predictive accuracy (%) (95%CIs)
50
40
30
20
10
0
200
1,000
2,000
4,000
Background tasks
No Refactoring

Lego structures
Tasks solved (%) (95%CIs)
35
30
25
20
15
10
5
0
200
1,000
2,000
4,000
Number of play tasks
No Refactoring

Refactoring improves performance

String transformations

Lego structures

Refactoring improves performance

String transformations

Lego structures

Refactoring reduces the size of the program

> Discovering abstractions before learning

> Discovering abstractions after/within learning

> Creating a smoother landscape for program induction

> Unifying high-level reasoning with perception

Inductive logic programming with entailment

Imagine learning a program for
the following image transformation

Inductive logic programming with entailment

Imagine learning a program for
the following image transformation

While learning,
we encounter two candidate programs
that result in the following outputs:

P_1:

P_2:

Inductive logic programming with entailment

Imagine learning a program for
the following image transformation

While learning,
we encounter two candidate programs
that result in the following outputs:

P_1:

P_2:

ILP techniques do not see that
P_1 is clearly better that P_2

Key idea: introduce an example-dependent loss function

Given	a set of positive examples	"Mr. James Bond" ↦ "Mr. Bond"
	a set of negative examples	"Mr. James Bond" ↦ "Mr. James"
	background knowledge	various string transformation functions
Find	a program that cover all positive and no negative examples	

Key idea: introduce an example-dependent loss function

Given	a set of positive examples	"Mr. James Bond" ↦ "Mr. Bond"
	a set of negative examples	"Mr. James Bond" ↦ "Mr. James"
	background knowledge	various string transformation functions
	an example-dependent loss function	f: $\epsilon \times \epsilon \rightarrow \mathbb{R}$
Find	a program that minimises the loss over positive examples	

Meet Brute: an ILP system with example-dependant loss

learning task

Invent a library of predicates

Search for the best library predicates to add to the program

Meet Brute: an ILP system with example-dependant loss

learning task

Invent a library of predicates

...

Search for the best library predicates to add to the program

Meet Brute: an ILP system with example-dependant loss

learning task

Invent a library of predicates

...

Search for the best library predicates to add to the program

add

to the program and search for a new library predicate

Experiments: Is Brute outperform entailment-based methods?

Robot planning

+ Manhattan distance

String transformations

+ Edit distance

"Mary Jane"

↧

"M. J."

ASCII images

+ Pixel distance

Experiments: Can Brute outperform entailment-based methods?

Our method

Robot planning

+ Manhattan distance

String transformations

+ Edit distance

ASCII images

+ Pixel distance

more complex problem

> Discovering abstractions before learning

> Discovering abstractions after/within learning

> Creating a smoother landscape for program induction

> Unifying high-level reasoning with perception

Humans interpret visual input in a structured way

Humans interpret visual input in a structured way

a dome

a really nice building

a tree

an entrance

Discovering visual abstractions

DeepProbLog: Unifying reasoning and neural perception

addition(, , Z) ←

digit(, X) , digit(, Y), Z = X + Y.

Manhaeve, Dumancic, Kimmig (Cardiff), Demeester (Ghent), De Raedt; NeurIPS 2018

Prolog and logic programming

Facts

edge(a,b).
edge(b,c).
edge(a,d).
edge(d,c).

Rules

path(X,Y) ← edge(X,Y).

path(X,Y) ← edge(X,Z),
path(Z,Y).

Prolog and logic programming

Facts

edge(a,b).
edge(b,c).
edge(a,d).
edge(d,c).

Rules

path(X,Y) ← edge(X,Y).

path(X,Y) ← edge(X,Z),
path(Z,Y).

Problog and probabilistic logic programs

Facts

0.9 edge(a,b).
0.6 edge(b,c).
0.7 edge(a,d).
0.7 edge(d,c).

Rules

path(X,Y) ← edge(X,Y).

path(X,Y) ← edge(X,Z),
path(Z,Y).

Problog and probabilistic logic programs

Facts

0.9 edge(a,b).
0.6 edge(b,c).
0.7 edge(a,d).
0.7 edge(d,c).

Rules

path(X,Y) ← edge(X,Y).
path(X,Y) ← edge(X,Z), path(Z,Y).

DeepProbLog: Deep learning + PLP

Facts

0.8 digit(,1).

0.01 digit(,2).

0.01 digit(,3).

…

Rules

```
addition(Im1, Im2, Sum) ←
        digit(Im1, X),
        digit(Im2, Y),
        Sum is X + Y.
```

DeepProbLog: Deep learning + PLP

Facts

0.8 digit(,1).

0.01 digit(,2).

0.01 digit(,3).

…

Rules

```
addition(Im1, Im2, Sum) ←
        digit(Im1, X),
        digit(Im2, Y),
        Sum is X + Y.
```

addition(, , S)

digit(, 1),
digit(, 3),
S = 1 + 3

digit(, 7),
digit(, 8),
S = 7 + 8

DeepProbLog: Deep learning + PLP

Facts

0.8 digit(,1).

0.01 digit(,2).

0.01 digit(,3).

…

Rules

addition(Im1, Im2, Sum) ←
digit(Im1, X),
digit(Im2, Y),
Sum is X + Y.

addition(, , S)

digit(, 1),
digit(, 3),
S = 1 + 3

digit(, 7),
digit(, 8),
S = 7 + 8

Evaluated with a NN

Neural predicate

The neural predicate represents the neural networks in the logic program

Uncertainty in predictions

Normalised output can be interpreted as probabilities

Discovering visual abstractions

DeepProbLog: Unifying reasoning and neural perception

addition(, , Z) ←

digit(, X) , digit(, Y), Z = X + Y.

Manhaeve, Dumancic, Kimmig (Cardiff), Demeester (Ghent), De Raedt; NeurIPS 2018

Discovering visual abstractions

DeepProbLog: Unifying reasoning and neural perception

addition(, , Z) ←

digit(, X) , digit(, Y), Z = X + Y.

Neural predicate:

Manhaeve, Dumancic, Kimmig (Cardiff), Demeester (Ghent), De Raedt; NeurIPS 2018

addition(, , 8)

addition(, , 8)

(a) 30 000 examples

(b) 3 000 examples

(c) 300 examples

addition(, , 8)

addition(, ,)

addition(, , 8)

addition(, ,)

addition(, , 14)

addition(, , 8)

addition(, ,)

addition(, , 14)

addition([,], [,], 64)

addition(, , 8)

addition(, ,)

addition(, , 14)

addition([,], [,], 64)

wap("Robert has 12 books. … How many does he have now?", 10)

addition(, , 8)

addition(, ,)

addition(, , 14)

addition([,], [,], 64)

wap("Robert has 12 books. … How many does he have now?", 10)

Induction by program sketching

Abstraction after learning
Abstraction before learning
Task 1
Task 2
Task 3
Task 4
Task 5
Abstracting visual information

Abstraction after learning
Abstraction before learning
Task 1
Task 2
Task 3
Task 4
Task 5
Abstracting visual information

Abstraction after learning
Abstraction before learning
Task 1
Task 2
Task 3
Task 4
Task 5
Abstracting visual information

Abstraction after learning
Abstraction before learning
Task 1
Task 2
Task 3
Task 4
Task 5
Abstracting visual information

Learning generative probabilistic programs

Learning generative probabilistic programs

```
def procedure(initial_conditions)
    def ray(start)
        def line(start)
        def line(collision)
    def mirror(location)
```