What can abstraction do for program induction

Sebastijan Dumančić
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 \times 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?

Task 1  Task 2  Task 3  Task 4  Task 5

Learning
Where does the abstraction fit?
Where does the abstraction fit?

Abstraction after learning

Task 1

Task 2

Task 3

Task 4

Task 5

Abstraction before learning
> 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?
Similarity in general knowledge representations?

- Features only
- Neighbouring features
- Structure only
- Random walks
- Proximity

Dumancic and Blockeel; MLJ 2017
Similarity in general knowledge representations?

- Features only
- Neighbouring features
- Structure only
- Random walks
- Proximity

2 jumps

Dumancic and Blockeel; MLJ 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?
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
1. extract candidate abstractions
1. extract candidate abstractions

2. re-represent the data
1. extract candidate abstractions
2. re-represent the data
3. learn to reconstruct
1. extract candidate abstractions

2. re-represent the data

3. learn to reconstruct

Dumancic, Meert, Guns, Blockeel; IJCAI 2019
1. extract candidate abstractions
2. re-represent the data
3. learn to reconstruct

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!

Cora-ER
AUC PR
original 0.18
latent 0.68

IMDB
AUC PR
.60
.78

WebKB
AUC PR
.66
.83

UWCSE
AUC PR
.29
.37

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?

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.
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.”
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!

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

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
Finding and choosing the abstractions

Initial program  Unfolded program

candidate

enumeration

...
Finding and choosing the abstractions

Initial program

Unfolded program

candidate

enumeration

folding

Alternative foldings
level 1
Finding and choosing the abstractions

Alternative foldings
level 1

folding

...
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

Objective:
minimise program size and redundancy
Does refactoring help to learn more efficiently?

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

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

Task 1  Task 2  Task 3  Task 4  Task 5

Refactor

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
---|---|---|---|---
Refactor | Refactor

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

Task 1
Task 2
Task 3
Task 4
Task 5

Refactor
Refactor

200 - 4000 tasks

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
Refactoring improves performance

String transformations

Predictive accuracy (%) (95% CIs)

Background tasks

Lego structures

Tasks solved (%) (95% CIs)

Number of play tasks
Refactoring improves performance

String transformations

Predictive accuracy (%) (95% CIs)

Background tasks

Lego structures

Tasks solved (%) (95% CIs)

Number of play tasks
Refactoring reduces the size of the program

String transformations

- Program size reduction (95% CIs)
- Background tasks
- Predicates (Functions)
- Literals (function calls)

Lego structures

- Program size reduction (95% CIs)
- Background tasks
- Predicates
- Literals
> Discovering abstractions before learning
> Discovering abstractions after/within learning
> Creating a smoother landscape for program induction
> Unifying high-level reasoning with perception
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:

ILP techniques do not see that \( P_1 \) is clearly better than \( P_2 \)
Key idea: introduce an example-dependent loss function

Given
- a set of positive examples
- a set of negative examples
- background knowledge

"Mr. James Bond" \mapsto "Mr. Bond"
"Mr. James Bond" \mapsto "Mr. James"

Find
- a program that cover all positive
  and no negative examples

various string transformation functions
Key idea: introduce an example-dependent loss function

Given
- a set of positive examples
- a set of negative examples
- background knowledge
- an example-dependent loss function

Find
- a program that minimises the loss over positive examples

"Mr. James Bond" $\mapsto$ "Mr. Bond"
"Mr. James Bond" $\mapsto$ "Mr. James"
various string transformation functions

$f: \epsilon \times \epsilon \to \mathbb{IR}$
Meet Brute: an ILP system with example-dependant loss

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

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

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

ASCII images
+ Pixel distance

“Mary Jane”
\[ \downarrow \]
“M. J.”
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 really nice building
- a dome
- a tree
- an entrance
Discovering visual abstractions

DeepProbLog: Unifying reasoning and neural perception

\[
\text{addition}(3, 5, Z) \leftarrow \\
\text{digit}(3, X), \quad \text{digit}(5, Y), \quad Z = X + Y.
\]
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).
**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).

Diagram:
- **path(a,c)?**
  - edge(a,c)
  - edge(a,b), edge(b,c)

  - edge(a,c)
    - ***✗***
    - edge(a,b), edge(b,c)
      - edge(a,b), edge(b,c)
        - ***✓***
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  \text{edge}(a,b).
0.6  \text{edge}(b,c).
0.7  \text{edge}(a,d).
0.7  \text{edge}(d,c).

Rules

\text{path}(X,Y) \leftarrow \text{edge}(X,Y).
\text{path}(X,Y) \leftarrow \text{edge}(X,Z), \text{path}(Z,Y).
DeepProbLog: Deep learning + PLP

Facts
0.8  digit(1,1).
0.01  digit(1,2).
0.01  digit(1,3).
...  

Rules
addition(Im1, Im2, Sum) ←
digit(Im1, X),
digit(Im2, Y),
Sum is X + Y.
DeepProbLog: Deep learning + PLP

Facts

- 0.8 digit(1, 1).
- 0.01 digit(1, 2).
- 0.01 digit(1, 3).
- ...

Rules

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

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

addition(1, 7, S)
  digit(1, 7),
  digit(3, 8),
  S = 7 + 8
DeepProbLog: Deep learning + PLP

Facts
0.8  digit(7,1).
0.01 digit(7,2).
0.01 digit(7,3).
...

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

digit(7, 1),
digit(3, 3),
S = 1 + 3

digit(7, 7),
digit(3, 8),
S = 7 + 8

addition(7, 3, S)

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( 3, 5, Z) ←
  digit( 3, X),  digit( 5, Y), Z = X + Y.
```
Discovering visual abstractions

DeepProbLog: Unifying reasoning and neural perception

addition( 3, 5, Z) ←

digit( 3, X), digit( 5, Y), Z = X + Y.

Neural predicate:
addition(5, 3, 8)
addition(5, 3, 8)
addition(5, 3, 8)

addition(5, 4, 9)
addition(5, 3, 8)
addition(5, 4, 9)
addition(5, 4, 14)
addition(5, 3, 8)

addition(5, 4, 9)

addition([3, 9], [2, 5], 64)

addition(5, 4, 14)
addition(5, 3, 8)
addition(5, 4, 9)
addition(5, 4, 14)
addition([3, 9], [2, 5], 64)

wap("Robert has 12 books. ... How many does he have now?", 10)
addition(5, 3, 8)

addition(5, 4, 9)

addition(5, 4, 14)

addition([3, 9], [2, 5], 64)

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

Induction by program sketching
Abstracting visual information before learning.

Abstraction after learning

Task 1  Task 2  Task 3  Task 4  Task 5

Abstraction before learning
Abstraction after learning

Abstraction before learning

Abstracting visual information

Task 1
Task 2
Task 3
Task 4
Task 5
Abstraction before learning

Task 1

Task 2

Task 3

Task 4

Task 5

Abstracting visual information

Abstraction after learning

Abstraction before learning
Learning generative probabilistic programs
Learning generative probabilistic programs

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