Deep Learning for Classical Planning

Michaela Urbanovská

21/01/2022

Outline

General intelligence

- H. Geffner's talk about Model-free learners and model-based solvers [5]
- Similar Kahneman's mind model with System 1 and System 2 in [7]
- To reach full potential we need both
- *In this presentation*
  - **model-based solver** automated planning
  - **model-free learner** deep learning

Introduction to Planning

- Problems typically modeled by hand
- Standard languages / representation (PDDL, PPDDL, STRIPS, FDR, ...)
- Solved by off-shelf planners

Introduction to Planning

Planning problem in PDDL

- Domain definition
  - Predicates
  - Actions - parameters, preconditions, effects
- Problem definition
  - Objects
  - Initial state - set of propositions
  - Goal state specification - set of propositions

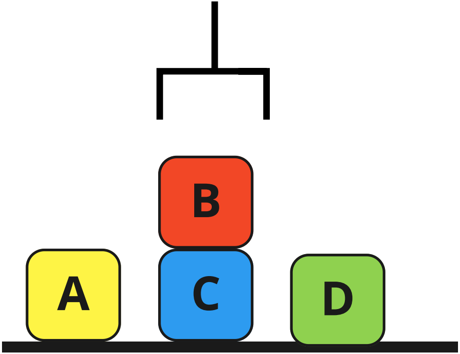
Introduction to Planning

```
(define (domain blocksworld)
  (:requirements :strips)
  (:predicates (on ?x ?y)
           (ontable ?x)
           (clear ?x)
           (handempty)
           (holding ?x)
           )

  (:action pick-up
         :parameters (?x)
         :precondition (and (clear ?x) (ontable ?x) (handempty))
         :effect
         (and (not (ontable ?x))
           (not (clear ?x))
           (not (handempty))
           (holding ?x)))

  (:action put-down
         :parameters (?x)
         :precondition (holding ?x)
         :effect
         (and (not (holding ?x))
           (clear ?x)
           (handempty)
           (ontable ?x)))
  (:action stack
         :parameters (?x ?y)
         :precondition (and (holding ?x) (clear ?y))
         :effect
         (and (not (holding ?x))
           (not (clear ?y))
           (clear ?x)
           (handempty)
           (on ?x ?y)))
  (:action unstack
         :parameters (?x ?y)
         :precondition (and (on ?x ?y) (clear ?x) (handempty))
         :effect
         (and (holding ?x)
           (clear ?y)
           (not (clear ?x))
           (not (handempty))
           (not (on ?x ?y)))))
```

```
(define (problem p01-blocksworld)
(:domain blocksworld)
(:objects A B C D)
(:INIT
     (ontable A) (ontable C) (ontable D) (on B C)
     (clear A) (clear B) (clear D) (handempty)
)
(:goal (AND
     (on C D) (on B C) (on A B)
     )
)
)
```



Introduction to Planning

Planning problem represented by STRIPS

$\Pi = \langle F, O, s_i, s_g, c \rangle$

- F - set of facts that can hold in the world
- O - set of operators which can be used to transform the world
- s_i - fully defined initial state of the world
- s_g - goal condition that holds in every goal state
- c - cost function which gives cost to every operator

Introduction to Planning

State

Every state $s \in S$ is a set of facts from F.

Operator

Every operator is a tuple that contains preconditions, add effects and delete effect for the given operator

$$o = \langle pre(o), add(o), del(o) \rangle$$

Operator o is applicable in state s if $pre(o) \subset s$. By applying o in s we get state s'

$$s' = (s \backslash del(o) \cup add(o))$$

Introduction to Planning

PDDL and STRIPS action

```
PDDL
(:action pick-up
      :parameters (?x)
      :precondition (and (clear ?x) (ontable ?x) (handempty))
      :effect
      (and (not (ontable ?x))
        (not (clear ?x))
        (not (handempty))
        (holding ?x)))

STRIPS
pickup(A) = <{clear(A), ontable(A), handempty},
            {holding(A)},
            {clear(A), ontable(A), handempty}>
```

Introduction to Planning

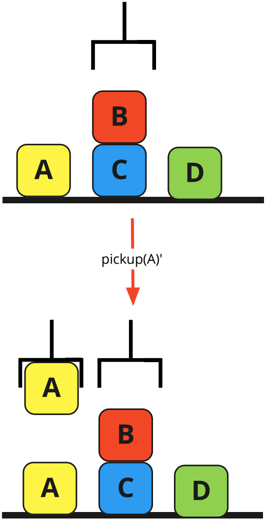
Relaxed STRIPS problem

- Simplification of the problem
- Delete-relaxation
- $\Pi' = \langle F, O', s_i, s_g, c\rangle$
- $o' = \langle pre(o), add(o)\rangle$

Introduction to Planning

```
PDDL
(:action pick-up
      :parameters (?x)
      :precondition (and (clear ?x) (ontable ?x) (handempty))
      :effect
      (and (not (ontable ?x))
        (not (clear ?x))
        (not (handempty))
        (holding ?x)))

Relaxed STRIPS
pickup(A) = <{clear(A), ontable(A), handempty},
             {holding(A)}>
```



Introduction to Planning

Transition system

$\Sigma = \langle S, A, \gamma, c \rangle$

- S - set of states
- A - set of actions
- γ - state transition function
- c - cost function

Solving a planning problem means looking for a path in the graph induced by the transition system.

- Forward search
- Backward search
- Bidirectional search

Introduction to Planning

Heuristic function

Heuristic function $h(s)$ maps any state s to a value that represents path length from s to a goal state.
Function that maps each state s to the length of shortest path from s to a goal is $h*$ which is the perfect or optimal heuristic.

Deep Learning in Classical Planning

- Many different possible applications (search, heuristic, grounding, policy...)
- Data which is not noisy
- Relatively small data sets
- Hard to compare with existing approaches

Deep Learning in Classical Planning

A lot of successful applications that have proved functionality

- Framework inspired by Kahneman's work [3]
- Learning policies and heuristics from images [6]
- Planning with images in latent space [1], [2]
- Using neural networks to learn heuristic functions [4]

Deep Learning in Classical Planning

Drawbacks of many of these approaches

- Input size or format
- Domain-independence / generalization abilities
- Size of the network
- Speed of the evaluation
- Time required for training
- Overall results

Deep Learning in Classical Planning

A couple approaches tried to create standardized architectures for planning purposes

- STRIPS-HGN - Hypergraph Neural Networks [8]
- ASNets - Action Scheme Neural Networks [10], [9]

STRIPS-HGN

- Works with a graph of the relaxed problem (STRIPS)
- Domain-independent
- *Input:* state - value pairs from optimal plans
- *Output:* domain-independent heuristic function represented by the HGN
- *Contributions*
    - Hypergraph framework which generalized GNNs (not main focus of the paper)
    - STRIPS-HGN architecture which is used for learning the heuristic functions
    - Evaluation that shows how STRIPS-HGN compares to relaxation heuristics (h^{max}, h^{add}, LM-cut)

HGN

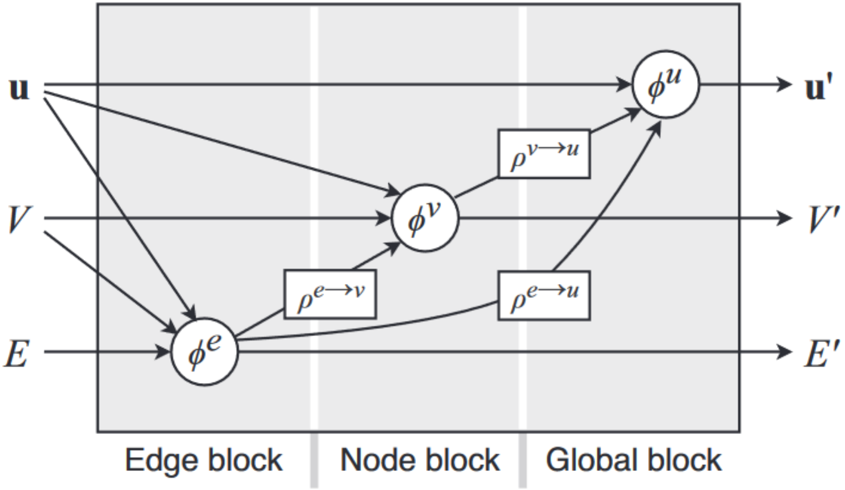
Hypergraph is defined as $G = (\mathbf{u}, V, E)$

- **u** - hypergraph (global) features
- V - set of vertices
- E - set of edges; each edge has features, list of "head" indices and list of "tail" indices

Hypergraph block

- Hypergraph to hypergraph function
- Contains **update** and **aggregation** functions
  - **Update** updates latent representation of vertices, hyperedges and global features (emulation of message passing)
  - **Aggregation** collect / pool features

HGN


u
V
E
u'
V'
E'
ϕ^u
ϕ^v
ϕ^e
$\rho^{v\to u}$
$\rho^{e\to v}$
$\rho^{e\to u}$
Edge block
Node block
Global block

HGN

- **Update** functions were implemented as MLP
- **Aggregation** functions were implemented as element-wise sum
  - should be permutation invariant

STRIPS-HGN

STRIPS-HGN is instantiation of HGN framework for learning heuristics.
STRIPS-HGN composes the HGN blocks into encode-process-decode architecture.

- *encode* block encodes input features into latent space
- *process* block is recurrently applied to the data to emulate message passing
- *decode* block obtains heuristic value from processed latent features

STRIPS-HGN

Input $G_{inp} = (\mathbf{u}_{inp}, V_{inp}, E_{inp})$ (hypergraph)

- follows structure of the relaxed STRIPS problem
- $\mathbf{u}_{inp}$ - global features (not required)
- V_{inp} - input features for $|F|$ propositions of the problem
  - true in current state
  - true in goal
  - fact landmark
- E_{inp} - hyperedges for all relaxed actions

Output $G_{out} = (\mathbf{u}_{out}, V_{out}, E_{out})$ (hypergraph)

- $\mathbf{u}_{out}$ - 1-dimensional vector that represents the heuristic
- V_{out} and E_{out} are empty sets

STRIPS-HGN

Encode block

- encodes the hypergraph into the latent space

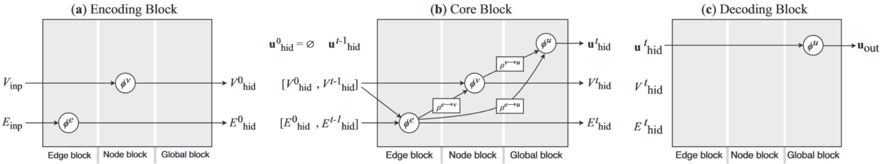
Process block

- in each step produces a new hypergraph
- hypergraph from previous iteration gets concatenated with the new one
- M times in total
- message passing M vertices away at maximum

Decode block

- decodes final hypergraph into G_{out} which has heuristic value in the global feature $\mathbf{u}_{out}$

STRIPS-HGN


(a) Encoding Block
V_{inp}
E_{inp}
ϕ^v
ϕ^e
V^0_{hid}
E^0_{hid}
Edge block
Node block
Global block
(b) Core Block
$\mathbf{u}^0_{hid} = \varnothing$ $\mathbf{u}^{t-1}_{hid}$
$[V^0_{hid}, V^{t-1}_{hid}]$
$[E^0_{hid}, E^{t-1}_{hid}]$
ϕ^u
ϕ^v
ϕ^e
$\rho^{v\to u}$
$\rho^{e\to v}$
$\rho^{e\to u}$
$\mathbf{u}^t_{hid}$
V^t_{hid}
E^t_{hid}
Edge block
Node block
Global block
(c) Decoding Block
$\mathbf{u}^t_{hid}$
V^t_{hid}
E^t_{hid}
ϕ^u
$\mathbf{u}_{out}$
Edge block
Node block
Global block

STRIPS-HGN - Training

Training data

- Set of training problems $P = \{p_1, p_2, ..., p_n\}$
- Solve every $p_i \in P$ and obtain h^* for every state on the optimal path
- Generate training pairs $(s, h^*(s))$
- Generate delete-relaxed hypergraph G for every p_i and s
- Get training samples $(G, h^*(s))$

STRIPS-HGN - Training

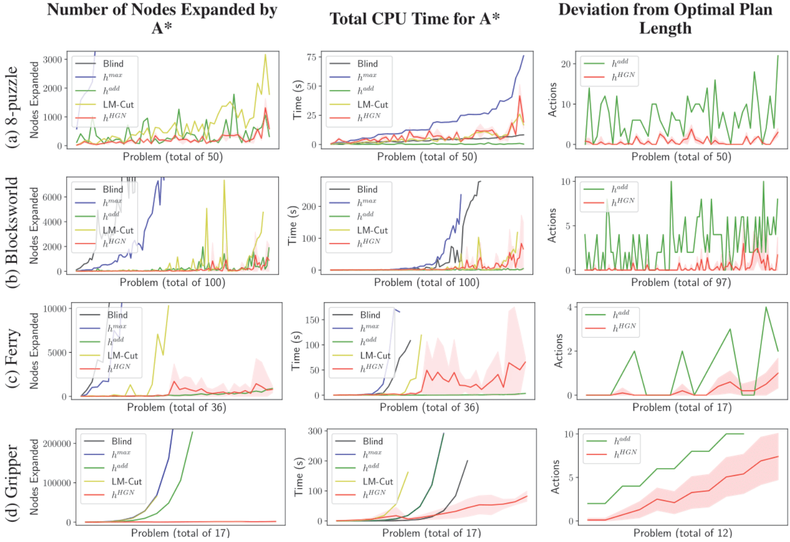
- *Process* block outputs a hypergraph in each step so loss function is aggregated over all steps
- MSE loss function
- Minibatch gradient descent
- Minibatch size $= 1$

STRIPS-HGN - Results

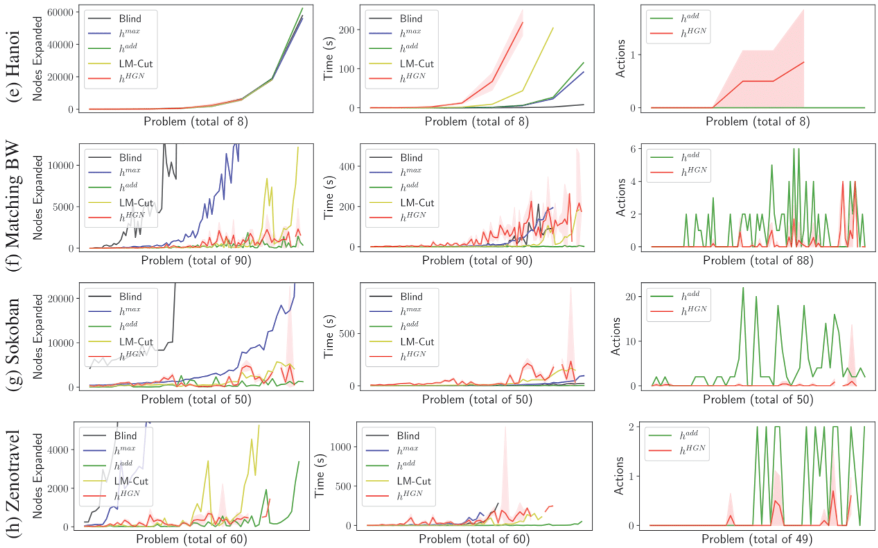
- 8 problem domains
- Different training configurations
  - Domain-specific (trained and tested on same domain)
  - Multi-domain (trained and tested on a set of 3 domains)
  - Domain-independent (trained on a set of domains, tested on unseen domains)

	blind	h^{max}	h^{add}	LM-cut	h^{HGN} spec.	h^{HGN} multi	h^{HGN} indep.
8-puzzle	1	1	1	1	1	–	–
Ferry	0.42	0.36	1	0.47	0.77	–	–
Hanoi	1	1	1	0.88	0.70	–	–
Mat. BW	0.85	0.85	1	0.98	0.83	–	–
Sokoban	1	1	1	0.96	0.91	–	–
BW	0.78	0.68	1	0.97	0.95	0.97	0.60
Gripper	0.71	0.59	0.59	0.41	0.95	0.69	0.29
Zeno	0.62	0.55	1	0.82	0.71	0.60	0.26

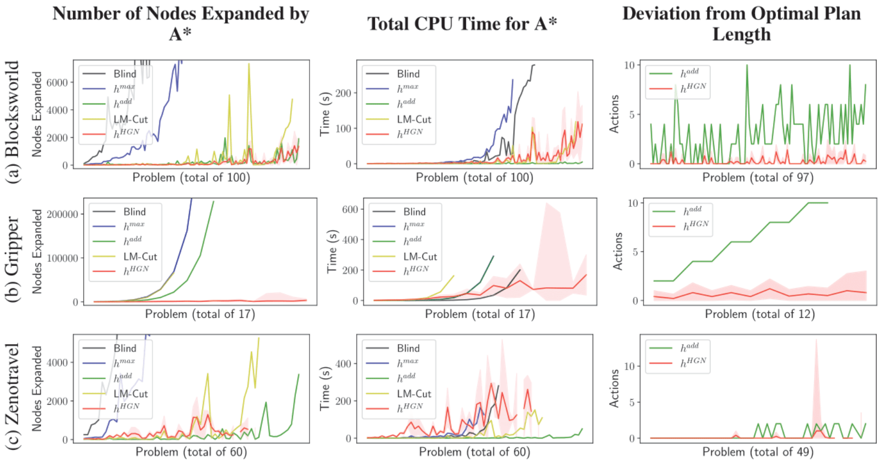
STRIPS-HGN - Results



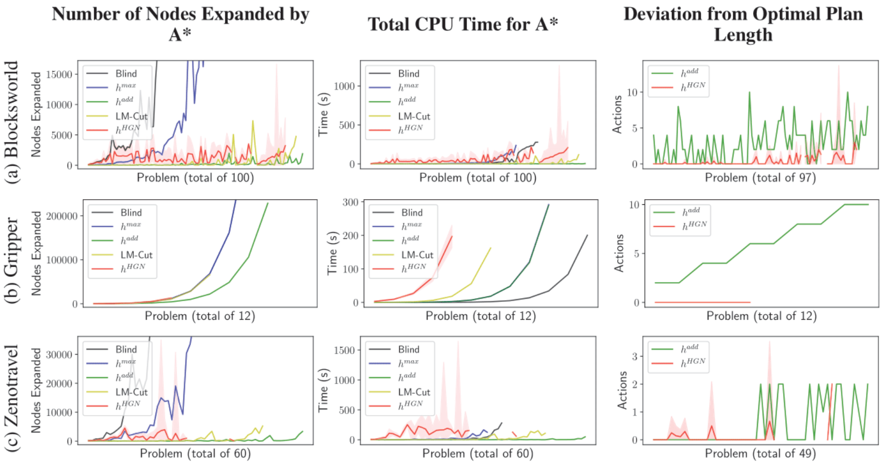
STRIPS-HGN - Results



STRIPS-HGN - Results



STRIPS-HGN - Results



STRIPS-HGN - Results

- Domain-specific
  - Quite impressive results
  - Better than some of the competing heuristics
- Multi-domain
  - Results showed ability to generalize over three training domains
- Domain-independent
  - Not very good performance
  - Problems with scaling (time)
  - Reusability of the learned knowledge visible in similar domains

STRIPS-HGN - Drawbacks

- Very expensive architecture to evaluate
  - Hard to use in a running search algorithm
- Tuning of parameter M is problematic
  - Increases time but gives better estimates
  - $M = 10$ in the experiments
- Parametrization based on number of receivers / senders on each edge
  - Uses padding in the feature vectors
  - Not "truly" domain-independent architecture

ASNets

- Learns generalized policy over planning problem (probabilistic or deterministic)
- Architecture mimics the problem definition scheme
- Weight sharing
- Not domain-independent
- *Input:* planning problem (probabilistic or deterministic)
- *Output:* trained network that provides a generalized policy for any problem from a given domain
- *Contributions:*
  - Architecture that generalized over any problem in given domain
  - Representation suitable for weight sharing
  - Training method for this architecture

ASNets - Background

Initially used on Stochastic Shortest Path problems (SSPs)
problem $P = (S, A, T, C, G, s_0)$

- S - set of states
- A - set of actions
- T - transition function ($T(s, a, s')$ probability of ending up in s' when selecting action a in state s)
- C - cost function
- G - set of goal states
- s_0 - initial state

Solution to SSP is a **policy** π

- π should aim to minimize expected cost of reaching G from s_0

ASNets - Background

Compact representation of SSP $\rightarrow$ *factored SSP* (P, A, s_0, s_*, C)

- P - set of binary propositions
- A - set of actions (each has preconditions and effects)
- s_0 - initial state
- s_* - goal state
- C - cost function

State space is defined as a set of all binary strings of length $|P|$.

ASNets - Background

Compact representation of set of factored SSPs $\rightarrow$ *lifted SSP* $(F, \mathcal{A}, C)$

- F - set of predicates
- $\mathcal{A}$ - set of action schemas
- C - cost function

PPDDL is standard language to describe lifted and factored SSPs.

- splits problem in to **problem** and **domain** definition

ASNets - Architecture

- Takes advantage of the action schemas
- Domain-specialized structure
- Using same set of weights θ with problem of any size (from one domain)
- Alternating **action layers** and **proposition layers**
- Initial version focused on PPDDL [10]
- Later more focus on PDDL [9]

ASNets - Architecture

Action layers

- Action layer l consists of **action modules**
- One module for one action schema A

$$\phi_A^l = f(W_A^l \cdot u_A^l + b_A^l), \phi_A^l \in d_h$$

- d_h is a fixed intermediate representation size

ASNets - Architecture

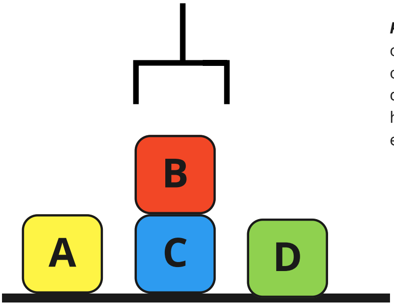
u_a^l (input) construction

- enumerate propositions $\{p_1, p_2, ..., p_M\}$ *related* to the action a
- p_i is related to a if p appears in $pre(a)$ or $eff(a)$
- concatenate their hidden representations from previous **proposition layer**
- ψ_j^{l-1} is hidden representation of p_j in the preceding **proposition layer** $l-1$

$$u_a^l = [\psi_1^{l-1} \dots \psi_M^{l-1}]^T, u_a^l \in d_h \cdot M$$

It is possible to use same weight matrix for every action instantiated from the action schema A.

ASNets - Architecture



Predicates

ontable(x)
on(x,y)
clear(x)
holding(x)
emptyhand

Propositions

ontable(A)
ontable(C)
ontable(D)
on(C,B)
clear(A)
clear(B)
clear(D)
emptyhand

Action schemas

pickup(x)
putdown(x)
stack(x,y)
unstack(x,y)

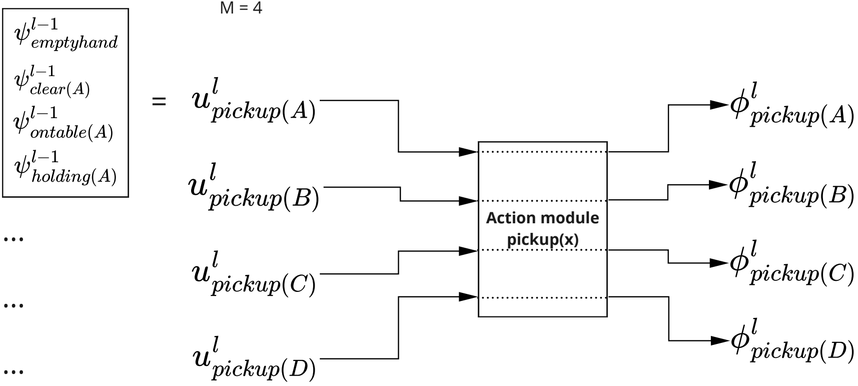
Actions

pickup(A)
...
pickup(D)
putdown(A)
...
putdown(D)
stack(A,B)
...
stack(D,C)
unstack(A,B)
...
unstack(D,C)

ASNets - Architecture

Action module - pickup(x)

related predicates = {emptyhand, clear(x), ontable(x), holding(x)}
M = 4



ASNets - Architecture

Input layer

- Classified as **action layer** with modifications

$$u_a^{in} = \begin{bmatrix} v \\ g \\ m \end{bmatrix}$$

- v - $v_i = 1$ if p_i is true in current state
- g - $g_i = 1$ if p_i is true in a goal state
- m - $m_i = 1$ if a is applicable in current state

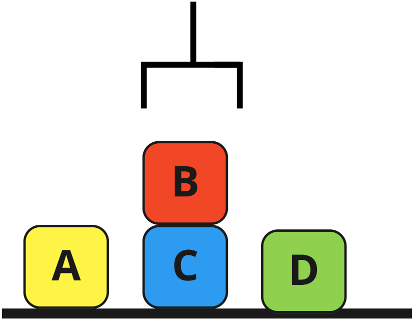
ASNets - Architecture

v = [1 1 1 0]'

$$g = \begin{array}{c} [\,0 \;\; 0 \;\; 0 \;\; 0\,]' \\ \begin{array}{cccc} \textit{emptyhand} & \textit{clear(A)} & \textit{ontable(A)} & \textit{holding(A)} \end{array} \end{array}$$

m = [1]

$u^{in}_{pickup(A)}$ = [1 1 1 0 0 0 0 0 1]'



ASNets - Architecture

Output layer

- Classified as **action layer** with modifications
- Outputs one digit for every action a that can be chosen in the current state
- Digits passed through masked softmax (only chooses from applicable actions)

ASNets - Architecture

Proposition layers

- Proposition layer l consists of **proposition modules**
- One proposition module for one predicate

$$\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l), \psi_p^l \in d_h$$

ASNets - Architecture

v_p^l (input) construction

- Enumerate all action schemas $A_1, ..., A_L \in \mathcal{A}$ which reference p in precondition or effect

$$v_p^l = \begin{bmatrix} pool(\{{\phi_a^l}^T | op(a) = A_1 \wedge R(a,p)\}) \\ \dots \\ pool(\{{\phi_a^l}^T | op(a) = A_L \wedge R(a,p)\}) \end{bmatrix}, v_p^l \in d_h \cdot L$$

- $pool(x)$ combines several d_h-dimensional vectors to one

ASNets - Architecture

Proposition module - clear(x)

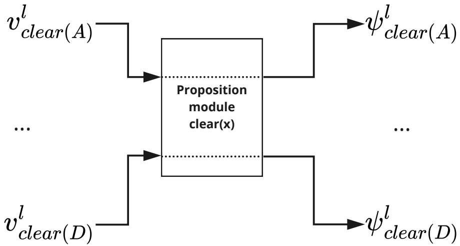
related action schemas = {pickup(x), putdown(x), stack(x,y), unstack(x,y)}
L = 4

$$\boxed{\begin{array}{l} \phi^{l-1}_{pickup(A)} \\ \phi^{l-1}_{putdown(A)} \\ pool(\{\phi^{l-1}_{stack(A,B)}, \phi^{l-1}_{stack(A,C)}, \phi^{l-1}_{stack(A,D)}\}) \\ pool(\{\phi^{l-1}_{unstack(A,B)}, \phi^{l-1}_{unstack(A,C)}, \phi^{l-1}_{unstack(A,D)}\}) \end{array}} = v^l_{clear(A)}$$

...

...

...



ASNets - Architecture

Architecture enhancements

- Different features can be added to the input
- Heuristic values, information about landmarks, ...
- Addition through extending the input feature vector of the first layer
- Skip connections

ASNets - Training

Algorithm 1 Updating ASNet weights θ using state memory $\mathcal{M}$ and training problem set P_{train}

procedure ASNET-TRAIN-EPOCH(θ, $\mathcal{M}$)
  for $i = 1, \ldots, T_{\text{explore}}$ **do** ▷ Exploration
    for all $\zeta \in P_{\text{train}}$ **do**
      $s_0, \ldots, s_N \leftarrow \text{RUN-POL}(s_0(\zeta), \pi^\theta)$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{s_0, \ldots, s_N\}$
      for $j = 0, \ldots, N$ **do**
        $s_j^*, \ldots, s_M^* \leftarrow \text{POL-ENVELOPE}(s_j, \pi^*)$
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{s_j^*, \ldots, s_M^*\}$
  for $i = 1, \ldots, T_{\text{train}}$ **do** ▷ Learning
    $\mathcal{B} \leftarrow \text{SAMPLE-MINIBATCH}(\mathcal{M})$
    Update θ using $\frac{d\mathcal{L}_\theta(\mathcal{B})}{d\theta}$ (Equation 1)

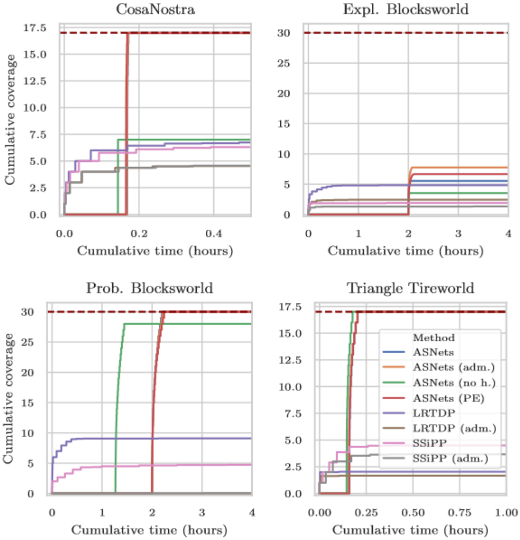
ASNets - Training

- Similar data collection as in bootstrapping methods
- *Loss function:* cross-entropy classification loss
- ADAM + SGD
- Other optimization methods tried
    - Cost of computing optimal policy for small samples is lower than optimizing with reinforcement learning
- 2 hours of training time

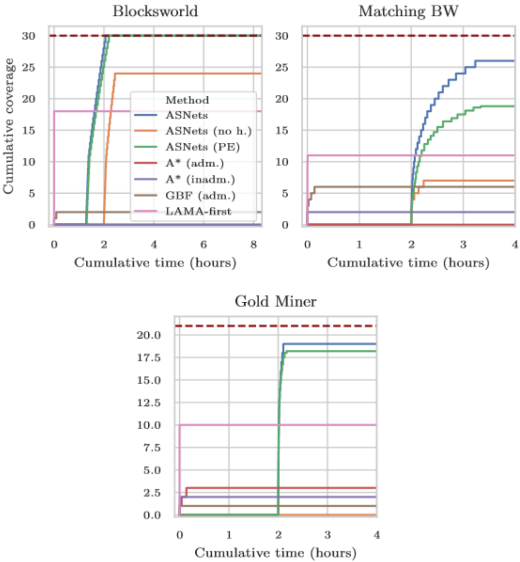
ASNets - Results

- Results for both probabilistic and deterministic domains (extended in related journal paper)
- One ASNet trained for each problem domain
  - All ASNets had the same architecture parameters
  - 3 action layers, 2 proposition layers, $d_h = 16$, ELU activation
- ASNet versions in the figures
  - ASNets -h^{add} teacher
  - ASNets (adm.) - admissible teacher (LM-cut heuristic features)
  - ASNets (no h.) - no heuristic inputs
  - ASNets (PE) - probabilistic execution (in the rollout)

ASNets - Results



ASNets - Results



ASNets - Results

Coverage of different architecture parametrizations (best possible coverage in **bold**)

Configuration	CN	ExBW	PBW	TTW	BW	GM	MBW
Default	**17.0/17**	5.5/30	**30.0/30**	**17.0/17**	**30.0/30**	19.0/21	26.0/30
Old-style pooling	**17.0/17**	4.5/30	**30.0/30**	**17.0/17**	**30.0/30**	**21.0/21**	28.0/30
No skip conn.	**17.0/17**	2.9/30	**30.0/30**	**17.0/17**	**30.0/30**	18.0/21	**30.0/30**
One layer	**17.0/17**	1.1/30	17.5/30	**17.0/17**	0.0/30	**21.0/21**	5.0/30
Three layers	**17.0/17**	**7.1/30**	**30.0/30**	**17.0/17**	**30.0/30**	19.0/21	**30.0/30**
No history	**17.0/17**	4.5/30	**30.0/30**	**17.0/17**	**30.0/30**	5.0/21	26.0/30
No LM-cut	**17.0/17**	5.2/30	**30.0/30**	**17.0/17**	**30.0/30**	13.0/21	25.0/30
No LM-cut/hist.	7.0/17	3.5/30	28.0/30	**17.0/17**	24.0/30	0.0/21	7.0/30

ASNets - Results

- Ability to generalize on larger instances after training on smaller ones
- Impressive convergence results in presented domains
- Starting point for more research in planning community applicable to different problems

ASNets - Drawbacks

- Line of reasoning is as long as number of layers (partially overcome by the added features)
- Quite expensive to evaluate
- Hard to scale architecture building fast on large problems (many actions and propositions)

Conclusion

- Deep learning in planning is still a hot topic
- Many works try to create a standard approach rather than reusing different tools for different problem
- Promising results
- A lot of drawbacks

Q&A

Thank you for your attention!

Questions?

Citations I

Masataro Asai and Alex Fukunaga.
Classical planning in deep latent space: From unlabeled images to PDDL (and back).
In Tarek R. Besold, Artur S. d'Avila Garcez, and Isaac Noble, editors, *Proceedings of the Twelfth International Workshop on Neural-Symbolic Learning and Reasoning, NeSy 2017, London, UK, July 17-18, 2017*, volume 2003 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

Masataro Asai and Alex Fukunaga.
Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary.
In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Di Chen, Yiwei Bai, Wenting Zhao, Sebastian Ament, John M. Gregoire, and Carla P. Gomes.
Deep reasoning networks: Thinking fast and slow.
CoRR, abs/1906.00855, 2019.

Hung-Che Chen and Jyh-Da Wei.
Using neural networks for evaluation in heuristic search algorithm.
In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

Hector Geffner.
Model-free, model-based, and general intelligence.
In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 10–17. ijcai.org, 2018.

Edward Groshev, Aviv Tamar, Maxwell Goldstein, Siddharth Srivastava, and Pieter Abbeel.
Learning generalized reactive policies using deep neural networks.
In *2018 AAAI Spring Symposium Series*, 2018.

Citations II

Daniel Kahneman.
Thinking, fast and slow.
Macmillan, 2011.

William Shen, Felipe Trevizan, and Sylvie Thiébaux.
Learning domain-independent planning heuristics with hypergraph networks.
In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 574–584, 2020.

Sam Toyer, Sylvie Thiébaux, Felipe W. Trevizan, and Lexing Xie.
Asnets: Deep learning for generalised planning.
J. Artif. Intell. Res., 68:1–68, 2020.

Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux, and Lexing Xie.
Action schema networks: Generalised policies with deep learning.
In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6294–6301. AAAI Press, 2018.