The background features a faded diagram of an Action Schema Network. It consists of a sequence of layers: 'Proposition layer 1', 'Action layer 2', 'Proposition layer L', and 'Action layer'. Each layer contains several rectangular nodes. Nodes in the 'Proposition' layers are blue, while nodes in the 'Action' layers are red. The nodes are interconnected by a network of lines, representing the schema's structure.

# Action Schema Networks: Generalised Policies with Deep Learning

Sam Toyer<sup>1</sup>, Felipe Trevizan<sup>1,2</sup>, Sylvie Thiébaux<sup>1</sup>, Lexing Xie<sup>1,3</sup>

<sup>1</sup>Australian National University, <sup>2</sup>Data61, <sup>3</sup>Data to Decisions CRC

# The piano mover's (other) problem

1. Send service robot to front door.
2. Make it push piano from front door to living room.



# The piano mover's (other) problem

1. Send service robot to front door.
2. Make it push piano from front door to living room.

Additional constraint: avoid obstacles which could bump/scratch piano.



# It's a factored probabilistic planning problem!

**Propositions:** state is an assignment to binary variables.

(at robot front-door): **false**,  
(at robot living-room): **true**,  
(undamaged piano): **true**, ...



# It's a factored probabilistic planning problem!

**Propositions:** state is an assignment to binary variables.

(at robot front-door): **false**,  
(at robot living-room): **true**,  
(undamaged piano): **true**, ...

**Actions:** move agent from state to state.

(push piano robot  
living-room  
hallway)

{ [(cluttered hallway)  
     $\Rightarrow p(\neg(\text{undamaged piano})) = 0.8]$   
   $\wedge$  (at piano hallway)  
   $\wedge$  (at robot hallway)  
   $\wedge \dots$

We must find a **policy** which chooses actions that (reliably) take us from the initial state to a goal state.



# Probabilistic Planning Domain Definition Language (PPDDL)

## Domain

A general “template” for a family of problems.

**Predicates:** lifted propositions; for instance:

(at ?object ?location), (path ?start ?end),  
(undamaged ?object), (cluttered ?location), ...

**Action schemas:** lifted actions; for example:

(push ?object  
  ?agent  
  ?from  
  ?to)  $\left\{ \begin{array}{l} [(cluttered\ ?to) \\ \Rightarrow p(\neg(undamaged\ ?object)) = 0.8] \\ \wedge (at\ ?object\ ?to) \wedge (at\ ?agent\ ?to) \wedge \dots \end{array} \right.$



## Instance

Describes one particular problem in a family of problems; associated with a single domain

**Objects:** piano, robot, front-door, ...

**Initial state:** (at piano front-door)  $\wedge$  (at robot living-room)  $\wedge$  (path living-room hallway)  $\wedge$  ...

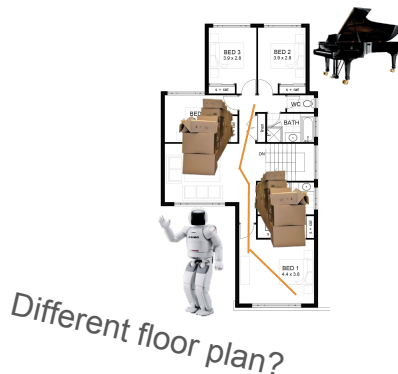
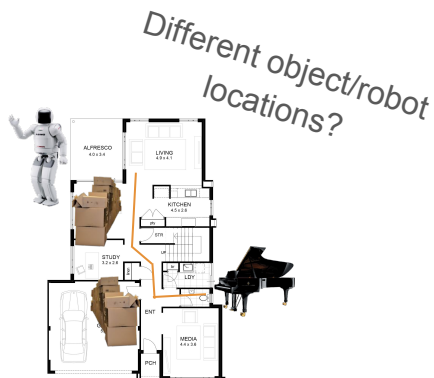
**Goal:** (at piano living-room)  $\wedge$  (undamaged piano)

Traditionally solved with a heuristic search planner.

# Generalised policies

A *generalised* policy can be applied to any problem in a domain.

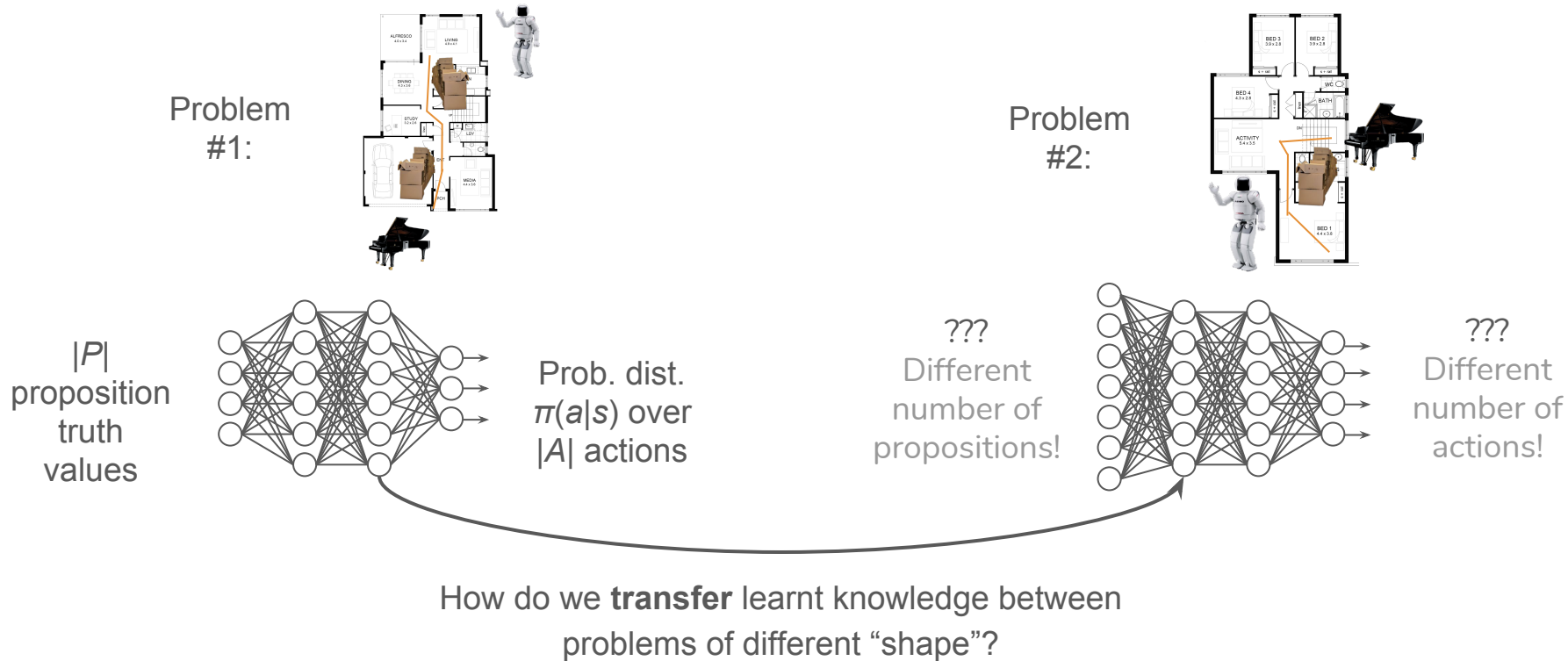
**Our contribution: generalised policies for probabilistic planning with neural networks.**



Traditional approach:  
Solve each problem in a set independently

Generalised policies:  
Learn the *1 weird trick* which allows you to  
easily solve *every* problem in a domain

# Generalised policies with NNs: first attempt





# Generalised policies with NNs: second attempt

**Big idea:** exploit structure by using separate network module for each action.

Only construct input from proposition which action affects or depends on.

Example for (move ?agent ?from ?to)

action schema:

(move	{	[(path ?from ?to)
?agent		$\wedge$ (at ?agent ?from)]
?from		$\Rightarrow$ [(at ?agent ?to)
?to)		$\wedge \neg$ (at ?agent ?from)]

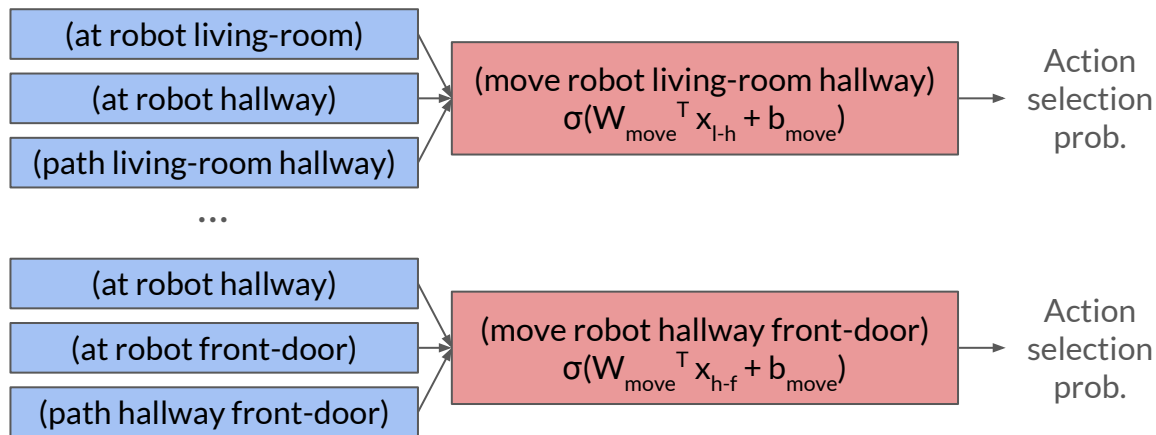
# Generalised policies with NNs: second attempt

**Big idea:** exploit structure by using separate network module for each action.

Only construct input from propositions which action affects or depends on.

Example for (move ?agent ?from ?to)  
action schema:

(move  
?agent  
?from  
?to) { [(path ?from ?to)  
           $\wedge$  (at ?agent ?from)]  
           $\Rightarrow$  [(at ?agent ?to)  
               $\wedge$   $\neg$  (at ?agent  
                      ?from)]



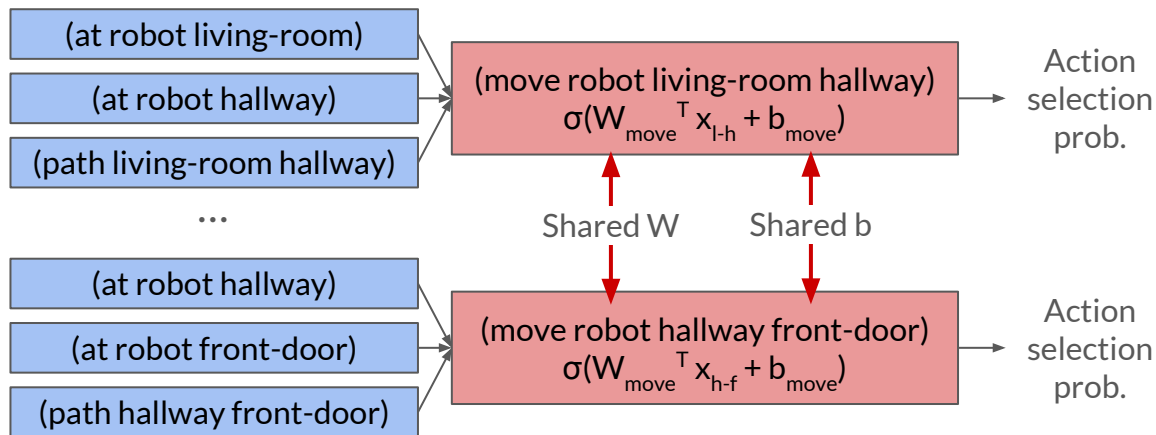
# Generalised policies with NNs: second attempt

**Big idea:** exploit structure by using separate network module for each action.

Only construct input from proposition which action affects or depends on.

Example for (move ?agent ?from ?to)  
action schema:

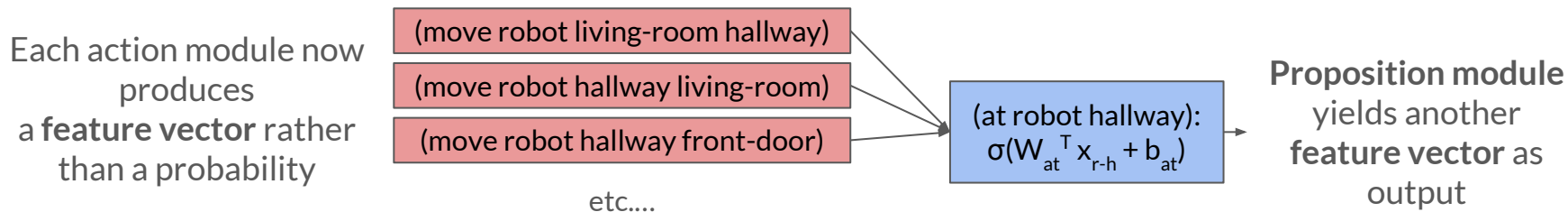
(move  
?agent  
?from  
?to) { [(path ?from ?to)  
           $\wedge$  (at ?agent ?from)]  
           $\Rightarrow$  [(at ?agent ?to)  
               $\wedge$   $\neg$  (at ?agent  
                      ?from)]



Actions with same schema now have same number of inputs, so we can share weights

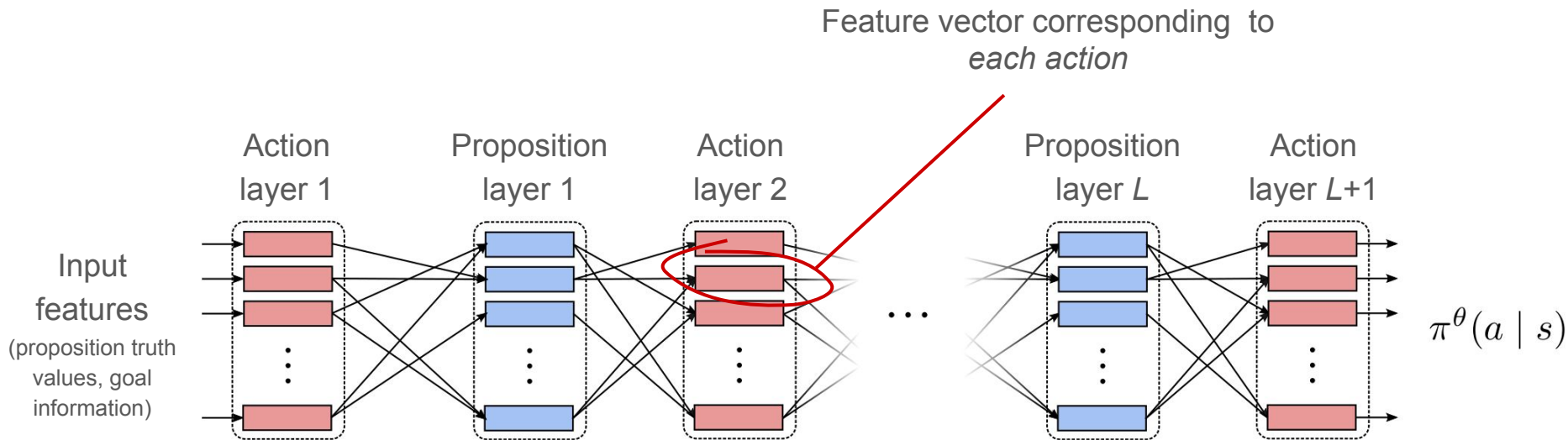
$\Rightarrow$  **allows us to generalise!**

# Going deeper with proposition modules



Proposition modules corresponding to same predicate have same input shape:  
we can **share weights** again.

# Action Schema Networks (ASNets)

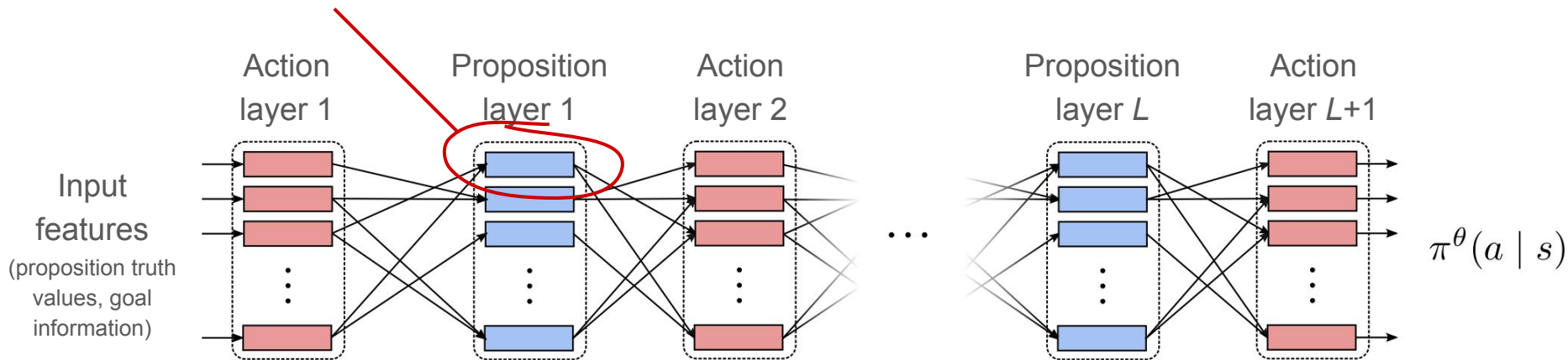


Transfer knowledge with **weight-sharing** between:

1. Action modules in same layer corresponding to same schema

# Action Schema Networks (ASNets)

Feature vector corresponding  
to *each proposition*



Transfer knowledge with **weight-sharing** between:

1. Action modules in same layer corresponding to same schema
2. Proposition modules in same layer corresponding to same predicate

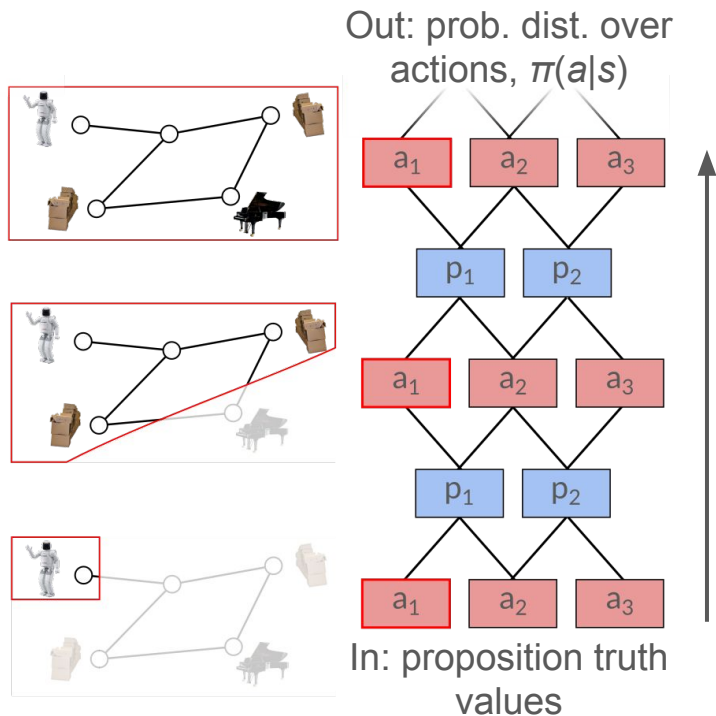
# Receptive field limitation

What if relevant propositions are outside receptive field?

⇒ Too shallow to “see” goal!

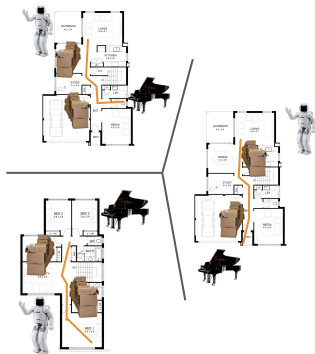
Solve by giving both **proposition truth values** *and* **heuristic-based inputs**.

In our case: indicator vector for each action module that tells us whether action is in a LM-cut landmark.



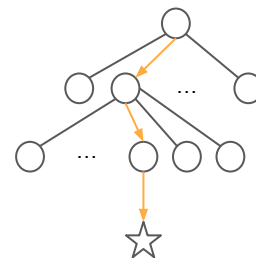
# Supervised training loop

1. Explore training environments with:  
(a) the learnt policy and  
(b) a “teacher” policy



3. Add observed  
(state, teacher Q-values)  
pairs to **memory  $M$**

2. Invoke teacher planner (LRTDP) to obtain **teacher Q-values** for each visited state and each action

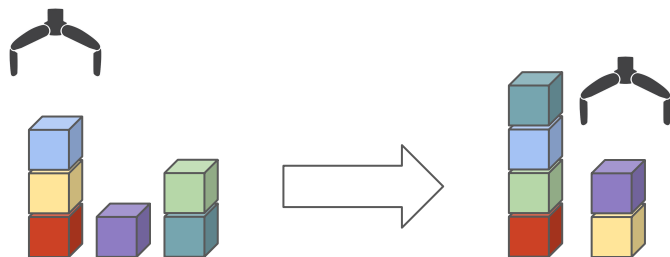


4. Sample pairs from  $M$  and take gradient descent step to **minimise action classification loss**

Policy should choose an action with **minimal** teacher Q-value  $Q^T(s,a)$



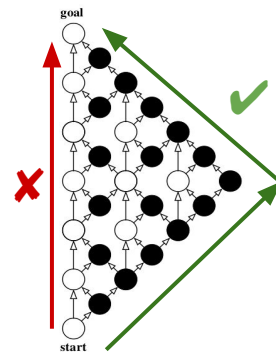
# Evaluation domains



Probabilistic Blocks World

25 training problems (with 5-9 blocks)

IPPC'08 domain (modified)



Triangle Tire World

3 training problems (with 6-28 locations)

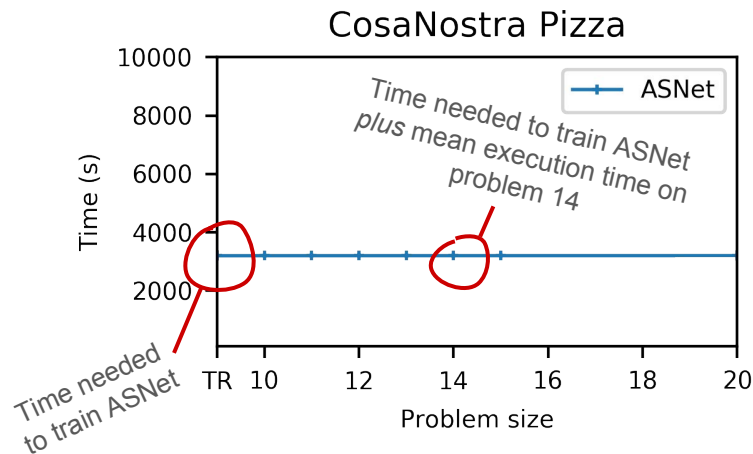
IPPC'08 domain



CosaNostra Pizza

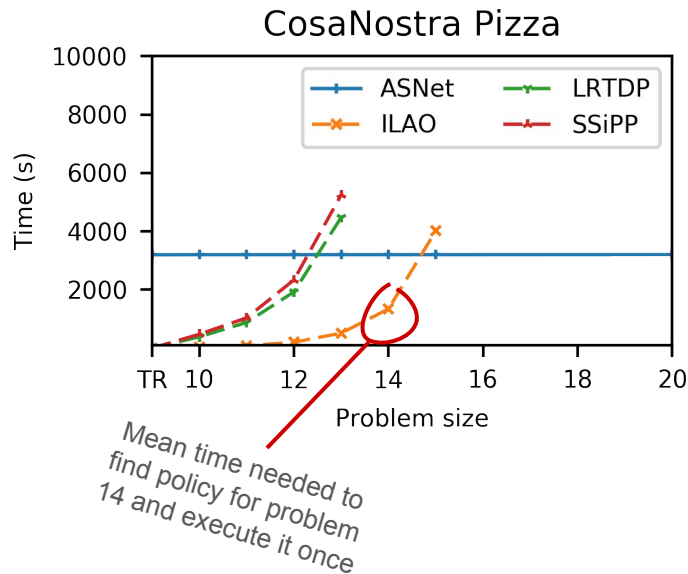
5 training problems (with 1-5 toll booths)

# Evaluation protocol



Evaluation emulates setting in which we only need our generalised policy to solve a **single** large problem.

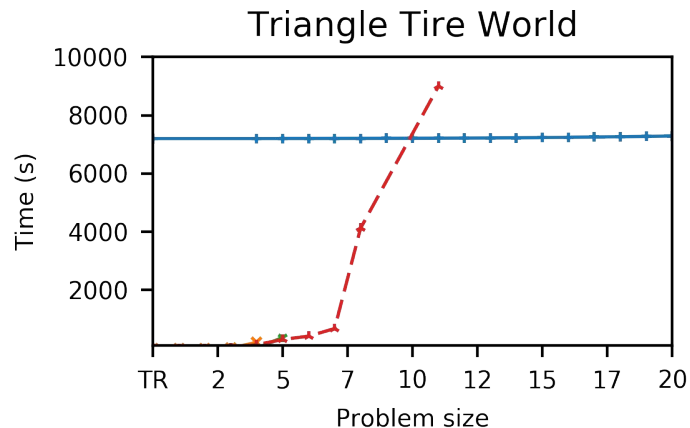
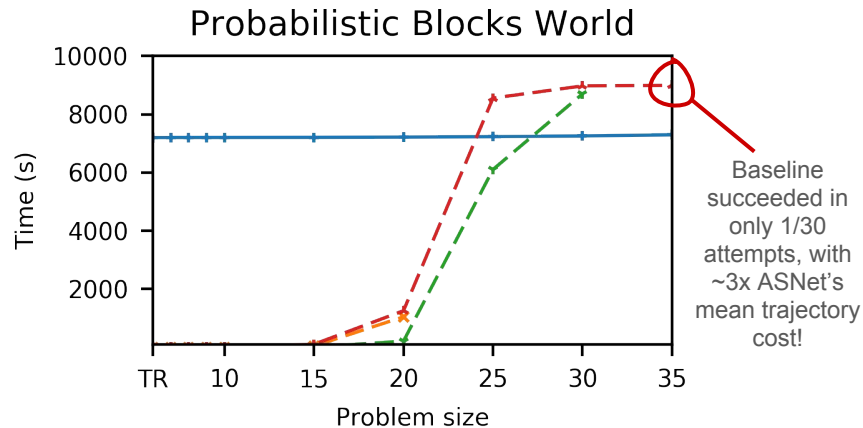
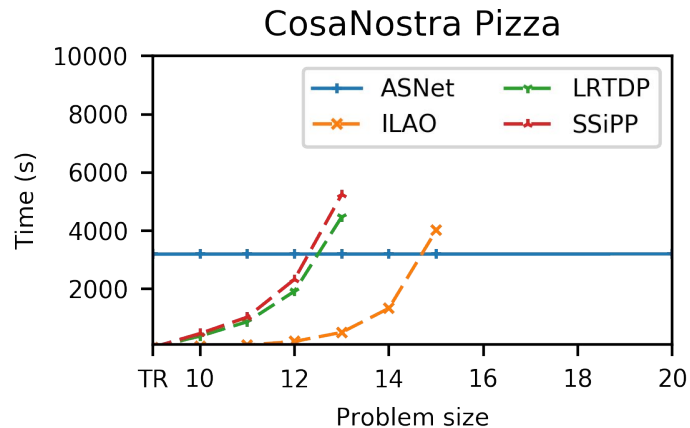
# Evaluation protocol



For baselines, we report **mean time for each baseline planner's value function to converge at  $s_0$ .**

Time limit of 9,000s (2.5h) on all planners.

Baselines in this presentation use  $h^{\text{add}}$  **heuristic**—LM-cut sometimes finds cheaper policies, but is far less scalable (see paper).



Works best for solving many problems, or very large problems, with a common “trick”.

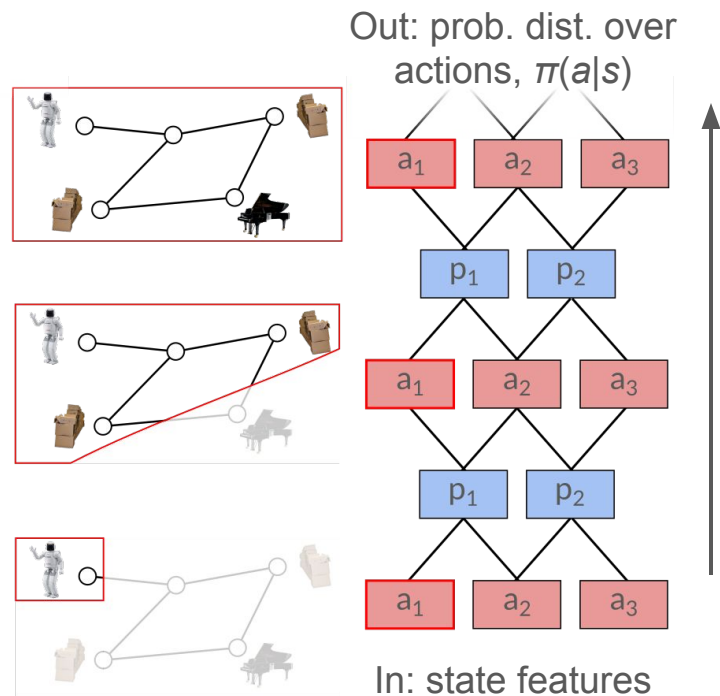
Learnt ASNet policy *a/ways* reaches the goal in these problems, with trajectory cost comparable to baselines—no optimality guarantees in general case, though!

# Summary and future work

- Our work: neural nets for generalised policies.
- Proposed **Action Schema Network (ASNet)**
- Generalised policies often **much faster** than planning one problem at a time

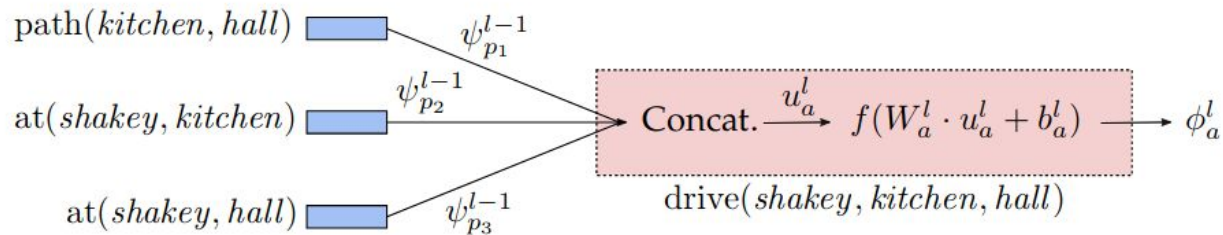
## Future work:

- Lift receptive field limitation
- Combining traditional search
- Explore representation learning, policy gradient RL, etc.

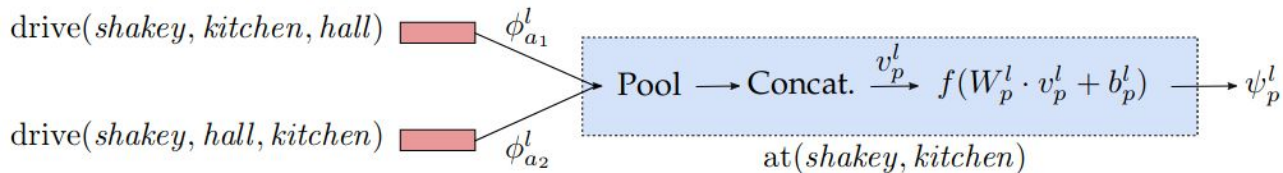


# Appendices

# Action and proposition modules

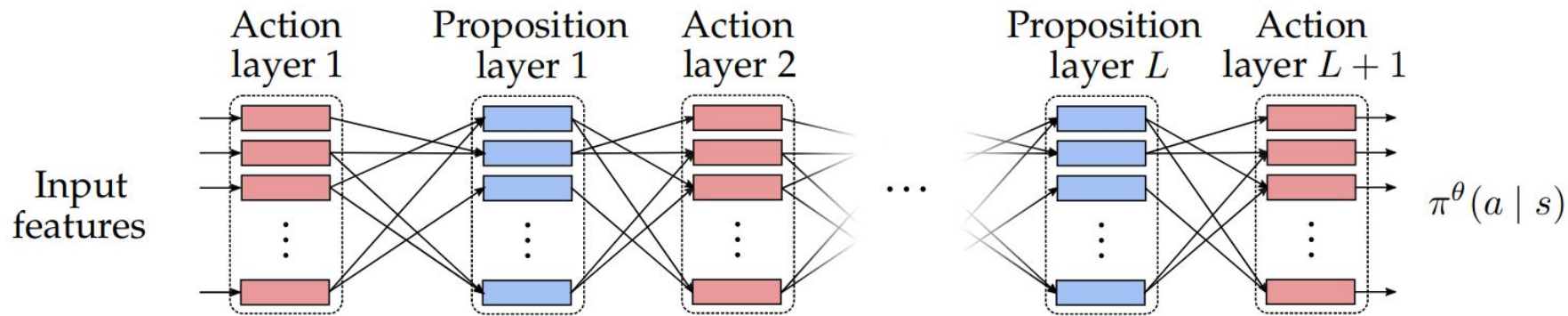


Action module



Proposition module

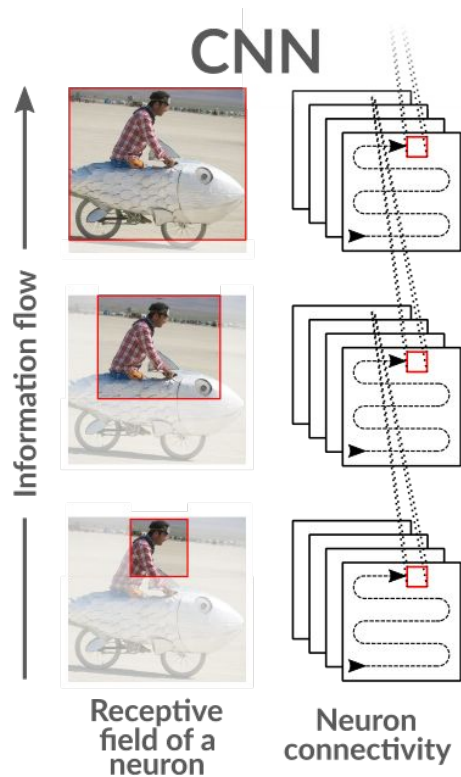
# Network structure



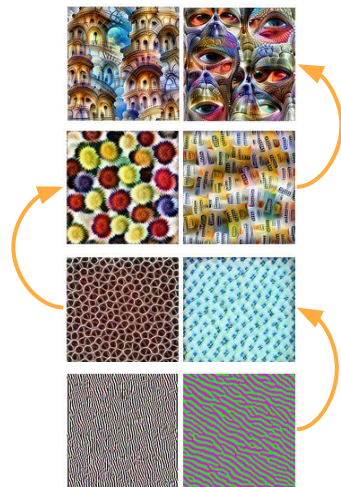
Layers wired up to propagate information between **related** action and proposition modules.



# Inspiration from convnets



Successively **more expressive** representations from **local operations** only.



# Related work

Several approaches to learning in **non-probabilistic** planning:

- Learn policies *or* heuristics
- Common representations: decision lists/trees

Three ways to improve:

1. Apply to probabilistic planning (minor)
2. Increase accuracy; flawed policies ruinous for planning!
3. More flexibility in loss

This work: build **flexible, accurate, generalised policy models** for **probabilistic problems**.

# Image credits

- Moving boxes: <https://udmlawconstruction.wordpress.com/page/3/>
- Assorted mess in hallway: <https://leasing.dmcihomes.com/deal-neighbor-problems-condominium/>
- “Smashed glass” floor covering:  
<http://rebloggy.com/post/room-mirror-glass-messy-hallway-corridor-broken-mirror-soft-grunge/56981800908>
- Grand piano:  
[https://commons.wikimedia.org/wiki/File:Steinway %26 Sons concert grand piano, model D-274, manufactured at Steinway%27s factory in Hamburg, Germany.png](https://commons.wikimedia.org/wiki/File:Steinway_%26_Sons_concert_grand_piano_model_D-274_manufactured_at_Steinway%27s_factory_in_Hamburg_Germany.png)
- Two-story floor plan: <https://www.apghomes.com.au/designs/under-350000>